2010

Development of a Flexible FPGA-Based Platform for Flight Control System Research

Robert DeMott
Virginia Commonwealth University

Development of a Flexible FPGA-Based Platform for Flight Control System Research

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

by

Robert C. DeMott II

Director: Dr. Robert H. Klenke
Associate Professor of Electrical & Computer Engineering

Virginia Commonwealth University
Richmond, Virginia
December 2010

Acknowledgements

This work would not have been possible without the help and support of many people. I wish to express my sincere appreciation to my advisor, Dr. Robert H. Klenke, for his support and guidance throughout the course of this project. I would also like to thank my thesis committee members, Dr. Mike McCollum and Dr. David Primeaux, for their feedback on my work. Thanks also go to the other faculty and staff of the Electrical and Computer Engineering Department as well as my fellow graduate students in the UAV Research Lab. In particular, I would like to thank Jeremy Cooper and Jose Ortiz for their friendship and assistance over the past few years. Finally, I am especially grateful to my family for their understanding, support, and encouragement during the time it took to complete my graduate studies.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADC	Analog-to-Digital Converter
AHRS	Attitude and Heading Reference System
AMSL	Above Mean Sea Level
API	Application Programming Interface
APU	Auxiliary Processing Unit
BGA	Ball-Grid Array
BRAM	Block Random Access Memory
CAN	Controller-Area Network
CAS	Calibrated Airspeed
CIC	Cascaded Integrator-Comb
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DC	Direct Current
DMA	Direct Memory Access
DMIPS	Dhrystone 2.1 MIPS
DSP	Digital Signal Processor
ECU	Engine Control Unit
EDK	Embedded Development Kit

EIA	Electronic Industries Alliance
EMI	Electromagnetic Interference
ENOB	Effective Number of Bits
ESL	Equivalent Series Inductance
ESR	Equivalent Series Resistance
FCM	Flight Control Module
FCS	Flight Control System
FIFO	First In, First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Simplex Link
FSM	Finite State Machine
GBP	Gain-Bandwidth Product
GCC	GNU Compiler Collection
GCS	Ground Control Station
GPIO	General Purpose Input/Output
GPS	Global Positioning System
I/O	Input/Output
IC	Integrated Circuit
ICM	Instrumentation Control Module
IIR	Infinite Impulse Response
IMU	Inertial Measurement Unit

INL	Integral Non-Linearity
INS	Inertial Navigation System
IP	Intellectual Property
ISE	Integrated Synthesis Environment
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LLA	Latitude, Longitude, Altitude
LSB	Least Significant Bit
LUT	Lookup Table
M2M	Module-to-Module
MAC	Media Access Control
MIPS	Million Instructions per Second
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
NED	North, East, Down
NMEA	National Marine Electronics Association
Op-Amp	Operational Amplifier
OPB	On-Chip Peripheral Bus
PCB	Printed Circuit Board
PHY	Physical Layer
PID	Proportional-Integral-Derivative
PLB	Processor Local Bus
POR	Power-on-Reset
PPTC	Polymeric Positive Temperature Coefficient

PSRR	Power Supply Ripple Rejection
PWM	Pulse Width Modulation
RAM	Random Access Memory
RC	Radio Controlled
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RMS	Root Mean Squared
RPM	Revolutions per Minute
RRIO	Rail-to-Rail Input/Output
RS-232	Recommended Standard 232
RTOS	Real-Time Operating System
SDRAM	Synchronous Dynamic Random Access Memory
SINAD	Signal-to-Noise and Distortion Ratio
SNR	Signal-to-Noise Ratio
SOM	System on Module
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
THD	Total Harmonic Distortion
TTL	Transistor-Transistor Logic
TVS	Transient Voltage Suppression
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus

VACS	VCU Aerial Communications Standard
VCU	Virginia Commonwealth University
VHDL	VHSIC (Very-High-Speed Integrated Circuit) Hardware Description Language
XCL	Xilinx Cache Link

Abstract

This work is part of ongoing research conducted at Virginia Commonwealth University relating to unmanned aerial vehicles. The primary objective of this thesis was to develop a flexible, high-performance autopilot platform in order to facilitate research on advanced flight control algorithms. A dual FPGA-based system architecture utilizing a stacked, multi-board design was created to meet this goal. Processing tasks were split between the two FPGA devices, allowing for improved system timing and increased throughput. A combination of analog and digital filtering techniques were employed in the new system, resulting in enhanced sensor accuracy and precision compared to the previous generation autopilot system. Several important improvements to the safety and reliability of the overall system were also achieved.

Chapter 1: Introduction

1.1 UAV Overview

The research and development of unmanned aerial vehicles (UAVs) for both military and civilian purposes has seen exceptional growth in the past decade. This increasing popularity is due to their many benefits over conventional, manned aircraft; the most obvious being the reduced risk to human life. This allows the aircraft to perform dangerous missions or execute maneuvers that would be harmful to a human pilot. Since there are is no crew aboard the aircraft, life support and associated safety systems are unnecessary. This can allow for a significant reduction in aircraft size and weight with a corresponding decrease in fuel requirements. These savings can translate into lower initial vehicle cost, as well as decreased operating cost and reduced maintenance requirements. In the case of small-scale unarmed UAVs, there is also less risk of collateral damage in the event of a crash.

Unmanned aerial vehicles are currently being employed in a diverse and ever expanding range of applications. In the military arena they are primarily used for reconnaissance, target acquisition and weapons delivery. In the civilian realm they are used for law enforcement, surveillance, search and rescue, crop dusting, and telecommunications. They are also used for remote sensing, weather monitoring and other forms of scientific research. This wide range of applications naturally results in an equally varied range of airframes, electronics, sensors, payloads, navigation systems, and flight control system (FCS) requirements.

1.2 Motivation

There are currently many autopilot platforms available for small UAVs, including both commercial systems and those developed at universities. Although some autopilots are designed to be fairly general purpose, many of these systems are very specialized and are only compatible with a limited range of airframes. They are often created to perform specific tasks, and are thus not always capable of being reconfigured for other applications. Another issue is that these systems are frequently based on older microcontrollers with limited storage and memory. Because of the lack the computational resources, these systems generally cannot handle complex control algorithms such as H-infinity loop-shaping or adaptive neural networks. They instead use simple linear feedback systems such as proportional-integral-derivative (PID) controllers. This limits the system's robustness and typically requires a lengthy tuning process for each airframe that the system is used with.

Virginia Commonwealth University's previous generation of flight control hardware has been successfully used in several different fixed-wing and rotary-wing aircraft for various purposes. Although it has been shown to be reasonably flexible, there are still some deficiencies that limit the system's capabilities and overall performance. The main issue is the system's limited processing power and memory bandwidth. This restricts the update rate of the current PID based flight control algorithms and prevents some types of sensors from being used due to overhead from data I/O and parsing. Like the systems mentioned above, this also prevents the FCS from implementing more complex control algorithms. Another issue is that the system lacks filtering of its analog input channels. This leads to noisy sensor readings and negatively impacts the flight control system response. These and other related issues provide the primary motivation for the design of a more powerful and capable autopilot platform.

1.3 Project Description

In this work, a flexible new hardware platform has been developed to overcome the limitations of the previous system and facilitate research of more optimal flight control algorithms. In the new system, two separate field programmable gate array (FPGA) based processor modules have been utilized. They have been dubbed the "Flight Control Module" (FCM) and the "Instrumentation Control Module" (ICM). The FCM is responsible for processing the flight control algorithms and handling higher-level autopilot functionality. This includes the navigation system and any mission specific requirements such as multi-UAV collaboration. The ICM is designed to handle all the lower level, I/O heavy tasks. This includes interfacing with all the necessary sensor devices, handling actuator control, and communicating with the ground station. In addition to interfacing with sensors, the ICM also parses all sensor data and converts it to a common format used by the FCM.

This division of labor between the two processor modules allows for a significant increase in system efficiency. In a typical single processor system, the CPU would have to handle sensor interfacing, parsing and actuator control routines in addition to the flight control and navigation algorithms. Also, many sensor interfaces such as RS-232 and SPI can generate a significant number of CPU interrupts. Not only do these interrupts directly contribute to overhead in the form of extra CPU cycles, but they can also result in costly cache misses. This can potentially cripple the throughput of an otherwise high-performance processor. By splitting the system into two modules, this situation can be avoided. As a result, complex control algorithms can execute at higher speeds and the system timing becomes more deterministic.

Since both modules (FCM & ICM) are based on FPGAs, custom user logic can be implemented in the hardware. In the case of the ICM, this programmable logic is primarily used

to implement the various interfaces needed by the sensors and actuators. This increases flexibility by allowing almost any sensor device to be connected simply by customizing the programmable logic. Custom logic is also used for high-speed digital filtering of analog sensor signals to reduce noise and provide cleaner data to the autopilot. In the case of the FCM, this programmable logic can be used to create an auxiliary processing unit (APU) and offload particularly complex instructions to accelerate the flight control algorithm.

The FCM processor is capable of running a real-time Linux-based operating system. This allows the application code for the flight control system to be easily debugged and developed more rapidly using standard tools. Also, since all communication between the two modules uses a single protocol, the FCM developer doesn't need to be concerned with the specific details of the individual sensor interfaces. Likewise, when adding support for a new sensor to the ICM, the specific details of the flight control software are irrelevant. These factors contribute to the ease of software development for this platform.

The physical hardware is also divided into two separate printed circuit boards (PCBs): the "main board" and the "auxiliary board". The main board contains components that will typically be required by all autopilots implemented on this platform. This includes the FCM and ICM, the power supply circuitry, several RS-232 serial interfaces, and the servo control & safety system, among other things. The auxiliary board is designed to be a low-cost, application specific add-on board. The current version of the auxiliary board contains analog sensor input and filtering circuitry, barometric pressure sensors, and connections for radio modems and GPS modules. Splitting up the hardware in this manner effectively increases flexibility by allowing the addition of new capabilities without requiring an expensive redesign of the entire system.

1.4 Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background information on previous work in the field of small UAV autopilots. This includes flight control systems developed through academic research as well as several commercially available systems. Chapter 3 presents the system hardware for the new autopilot platform. This includes a discussion of the FPGA-based processor modules chosen and the design of the printed circuit boards and related components. Chapter 4 describes the custom FPGA user logic that was created and utilized in the ICM hardware. Chapter 5 covers the ICM software and the inter-module interface protocol. Chapter 6 analyzes the overall performance achieved by the complete system. Chapter 7 summarizes the results of this work and provides suggestions for future improvements.

The development of the FCM software, including implementation of the flight control algorithms and the porting of a real-time capable operating system were handled by another graduate student and are not covered in this thesis.

Chapter 2: Background and Previous Work

Information on prior work related to autopilot systems for small UAVs will be provided in this chapter. Previously developed systems, including those that are currently available on the market, as well as systems still under active development will be covered. A description of each system's hardware platform will be provided. This will include its processing power, sensor configuration and communications sub-system. The software capabilities, including information on the flight control and navigation algorithms utilized will also be provided if available. System flexibility and aircraft compatibility will also be mentioned.

First, earlier research carried out at Virginia Commonwealth University will be discussed. Current commercial off the shelf autopilots will be covered next, followed by several research level autopilot systems designed at other universities and institutions. Finally, a comparison of the hardware used by the various autopilots will be presented.

2.1 Past Research at Virginia Commonwealth University

Virginia Commonwealth University has conducted both graduate and undergraduate research involving unmanned vehicle control systems for the past several years. The previous version of VCU's autopilot, known as the "Suzaku FCS", has proven to be a fairly versatile and capable system. It was originally designed for use with small fixed-wing UAVs, but it has since been adapted for use with rotary-wing aircraft. A detailed description of this system, including its features and limitations, will be provided in the following sections.

2.1.1 Autopilot Hardware

The hardware for the Suzaku FCS is based on the Suzaku-S processor board from Atmark Techno. The Suzaku-S includes a Xilinx Spartan-3 FPGA, 16MB of SDRAM, 8MB of Flash and a 100Mbps Ethernet interface. A 48 MHz MicroBlaze soft-core processor with a floating point unit (FPU) is implemented in the FPGA using standard logic resources and runs a light-weight version of the Linux operating system, known as μClinux [1, 2, 3].

A custom PCB was used to integrate the Suzaku-S module with the rest of the system. This board, shown in Figure 2.1, includes static and dynamic barometric pressure sensors, a 4 Hz uBlox GPS module, a 900 MHz MaxStream radio modem, an 8 channel 16-bit analog-to-digital converter, and a large number of digital and analog I/O pins. FPGA resources in the Spartan-3 were used to implement the various sensor interfaces, including several RS-232 UARTs and an SPI bus. Peripheral cores were also created for monitoring the PWM signals from the standard RC safety pilot receiver, and generating the PWM signals for controlling the aircraft's control surface actuators. Various external sensors for measuring attitude information have been utilized based on the particular aircraft and the accuracy required. Examples include the CoPilot Horizon Sensors from FMA, the Microstrain 3DM-GX1 IMU, and the Microbotics MIDG-II INS [1, 2].

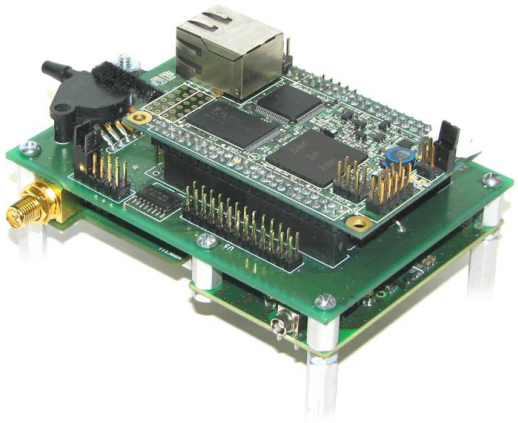

Figure 2.1: Suzaku FCS Hardware

2.1.2 Autopilot Software

The Suzaku FCS software is written in C and runs as a process in the μClinux operating system. The fixed-wing version of the FCS algorithm runs at 25 Hz and utilizes several cascaded PID controllers with gain scheduling and feed-forward components. The system supports multiple navigation modes and allows for parameters and waypoints to be changed in real-time. This autopilot has been flown in numerous airframes, including modified FQM-117B "Mig 27" target drones, the DV8R and Jurassic turbine-powered RC Jets, and the Outlaw UAV from Griffon Aerospace [1].

The rotary-wing version of the software shares many of the same features of the fixed-wing FCS and adds support for a gimbaled camera system. Different control laws were necessary, however, due to the unique handling properties of rotary-wing aircraft. Also, the control loops are updated at 50 Hz in order to compensate for the aircraft's inherent instability. This autopilot system has been used primarily in electric powered RC helicopters, such as the X-Cell Ion-X [1, 4].

2.1.3 Autopilot Variations

A variant of the fixed-wing autopilot was used in NASA's Flying Controls Testbed (FLiC). The FLiC was a small, inexpensive unmanned aircraft that was modified to significantly increase the number of control surfaces. The main goal of the project was to allow for testing of highly experimental flight control methods developed using MATLAB/Simulink. In this system, the Suzaku-S processor module used in the normal autopilot was replaced with the more powerful Suzaku-V. This module utilized a Xilinx Virtex-II Pro FPGA with an embedded PowerPC 405 processor running the Linux 2.6 Kernel. It also had 32MB of SDRAM, 8MB of Flash, and a 100Mbps Ethernet interface [5, 6].

The VCU autopilot hardware was also used in NASA's Generic Transport Model (GTM) Project. The GTM project conducted experiments using dynamically scaled models of passenger transport jets to study their behavior outside of the typical flight envelope. In this project, the Suzaku-S hardware was used in both the standard configuration described previously as well as in a modified form that functioned as a sensor data acquisition system. The data acquisition system combined the Suzaku-S processor module with a custom sensor interface board that featured 32 analog input channels and 18 PWM signal monitoring channels in addition to the typical serial ports and GPIO pins. The PWM input channels were used to record the manually commanded servo control signals, while the analog inputs were primarily used to record the actual positions of the aircraft's control surfaces using potentiometers. The system also integrated barometric pressure sensors for capturing altitude and airspeed information and an external Microbotics MIDG-II INS for position and orientation measurement.

2.1.4 Autopilot Limitations

As mentioned previously, the Suzaku FCS has performed well and was used effectively in many configurations. Nevertheless, it does suffer from several limitations, most of which stem from hardware issues with the Suzaku-S module and the system main board. Although most of the problems have existed since the system was originally designed, some of them have only recently become limiting factors. A few were only discovered after the capabilities of autopilot software increased. These issues will be listed and described in detail below. Later chapters will discuss how these issues are addressed by the new autopilot hardware designed for this thesis.

2.1.4.1 Limited Processing Power

The MicroBlaze soft-core processor implemented in the Suzaku-S FPGA is theoretically capable of achieving over 44 DMIPS (Dhrystone 2.1 MIPS) when running at 48 MHz, an efficiency of 0.92 DMIPS per MHz. However, in practice it only manages about 8.5 DMIPS, or less than 20% of its maximum performance [2]. This is primarily the result of two problems with the Suzaku-S module.

The first issue concerns the SDRAM interface used by the Suzaku-S module. The SDRAM controller implemented in the FPGA expects that the SDRAM be directly connected to the FPGA using a dedicated bus. The Suzaku-S instead shares the SDRAM memory bus with the Ethernet controller, parallel Flash memory, and JTAG interface IC [3]. Atmark Techno uses custom logic in the FPGA to essentially multiplex these devices and as a result the SDRAM controller must run at significantly reduced performance in order to function correctly. In addition, the SDRAM controller connects to the MicroBlaze over the On-Chip Peripheral Bus which is also used by peripheral cores such as SPI and UART controllers. Memory throughput is thus further reduced due to bus arbitration.

The second issue is that the MicroBlaze data cache must be disabled to ensure proper operation. Enabling the data cache does improve performance; however it also results in intermittent failure of the autopilot software. This appears to be the result of a bug in the μClinux distribution provided by Atmark Techno which is based on a release from 2005. Fixing this bug would likely require a significant time investment and would only partially remedy the performance issues.

The lack of a data cache in combination with the slow memory bus can result in delays of over 50 clock cycles per word when transferring data between the MicroBlaze and SDRAM.

Because of these limitations, the current autopilot software consumes almost all of the available CPU time, leaving insufficient computational resources for testing more complex algorithms or handling additional sensors.

2.1.4.2 Analog Noise Issues

The custom main board used for the Suzaku FCS does not have anti-aliasing low-pass filters on any of its analog inputs. This means that any frequency content higher than half of the sampling rate (Nyquist Frequency) will be reflected (aliased) into the bandwidth of the sampled signal, resulting in distortion. The altitude and airspeed data from the barometric pressure sensors are particularly affected by this problem. This is partially due to the fact that the pressure transducers are susceptible to vibration, often resulting in tens of millivolts worth of noise. In the case of the altitude measurements, this can translate to rapid variations of greater than ±20 feet. This measurement noise can result in the FCS making excessive control surface corrections, which degrades flight performance and increases mechanical wear.

The above issue with analog noise is further exacerbated due to board layout. There is almost no isolation between the analog and digital portions of the main board. Because of this, noise from switching power supplies and high speed digital nets can easily couple into the analog inputs. The ripple voltage from the 3.3V switching regulator is particularly problematic due to the regulator's close proximity to the ADC.

Attempts to mitigate these issues through software means, such as oversampling and digital filtering, were only partially successful. The use of oversampling was limited due to the high number of CPU cycles needed to transfer data from the ADC to the Suzaku-S via the SPI bus. Thus, only a small number of redundant samples could be taken, resulting in negligible noise reduction. A software based moving average filter was also used to reduce noise by

combining samples from the past several FCS updates. Unfortunately, this added significant delay to the altitude and airspeed measurements which made it more difficult to tune the respective PID control loops. If the gains were set too high there would be noticeable overshoot and oscillations in the control response. However, lower gains would result in a more sluggish response to set-point changes.

2.1.4.3 Wireless Data Link

The wireless link used by the Suzaku FCS is based on the 900 MHz XStream series of radio modems from Digi-MaxStream. These modems are essentially wireless RS-232 serial links running at either 9600 baud or 19200 baud. This translates to a maximum throughput of 960 or 1920 bytes per second, respectively. Throughput is somewhat lower in practice due to the half-duplex nature of the modems and the overhead associated with switching between transmit and receive. Due to the low data rate, the frequency of status updates from the UAV is limited to a maximum of 2 to 4 Hz. The amount of data per update is also significantly restricted.

2.1.4.4 Non-Volatile Storage

The Suzaku FCS hardware also lacks any substantial amount of onboard, non-volatile storage. This means that any data obtained during flight must either be transferred to the ground station via a wireless link or stored in memory and retrieved after flight. Because the wireless link is fairly slow, only a small subset of the data can be transferred in this manner. Instead, data recorded during flight is stored to a log file located in SDRAM. Operating system overhead limits this file to a maximum size of 8 MB, or roughly 15 minutes worth of flight data. The main problem with this approach is the volatile nature of SDRAM. If the autopilot loses power before

the log file is retrieved, all recorded data will be lost. This would be particularly troubling in the event of a crash, since the log file would have been useful in determining the cause of failure.

2.1.4.5 Power Protection Circuitry

Another issue with the Suzaku FCS main board is the lack of any protection circuitry at the power input. This protection circuitry would normally serve to prevent over-voltage or reverse-voltage conditions from damaging the system. These situations may arise during testing if a lab power supply is improperly configured or if a battery is accidently connected backwards. Such user error has resulted in the destruction of Suzaku-S modules on more than one occasion.

2.1.4.6 External Safety Switch

In the previous system, an external safety switch was used to allow for switching of aircraft control between the autonomous outputs from the FCS and manual control by a human pilot. This functionality is a necessary fail-safe that allows for manual override in the event of an autopilot failure. The use of external hardware for this function has both advantages and disadvantages. On the positive side, a reliable and well tested safety switch will remove a critical point of failure from the autopilot and theoretically provide a more consistent degree of safety regardless of the specific autopilot hardware being tested.

On the other hand, a discrete safety switch is less convenient than an integrated solution and can actually increase the number of potential failure points. A stand alone switch reduces payload capacity, both in terms of physical space and weight, and it also increases the complexity of the control signal wiring in the form of additional cable splitters and extension segments. This added wiring is needed to route the manual control signals from the RC receiver to both the safety switch and to the autopilot. This increases the number of connectors in the

signal path; and with each connector there is the risk of it coming loose during flight or causing intermittent issues due to poor contact quality.

Finally, one additional issue concerns the particular safety switches that were used in the past. Under certain high load conditions, these switches could enter a failure mode that resulted in rapid switching between manual and autonomous control. This was due to voltage fluctuations triggering a reset in the microcontroller used to control the multiplexer circuitry. If this were to happen during flight, it would almost certainly result in a crash.

2.2 Commercial Autopilots

2.2.1 MicroPilot: MP Series

MicroPilot has created several commercial autopilot systems for use with small UAVs. The MP2028^{g} is MicroPilot's mid-range autopilot solution for fixed-wing aircraft, while the MP1028^{g} is their low-end version. The MP2128^{g} and MP2128^{HELI} are MicroPilot's high-end offerings for fixed-wing and rotary-wing UAVs, respectively. The MP2028^{g} and MP2128^{HELI} will be covered in more detail below.

The MP2028^{g}, shown in Figure 2.2, is a single board system that measures 10.0 cm in length by 4.0 cm in width and weighs 28 grams. The system is based around a low power Motorola 68332 microcontroller and has 1.5 MB of memory dedicated to data logging. The onboard sensor system integrates a 1 Hz GPS receiver, 3-axis accelerometer, 3-axis gyroscope, and barometric pressure sensors for altitude and airspeed information. An optional 3-axis magnetometer can also be added to provide improved heading information. The communications system can utilize a 900 MHz or 2.4 GHz radio modem from MaxStream or Microhard and allows for a telemetry update rate of 5 Hz. A maximum of 24 servos are supported and can be updated at a rate of 50 Hz with 11-bit pulse-width modulation (PWM) resolution.

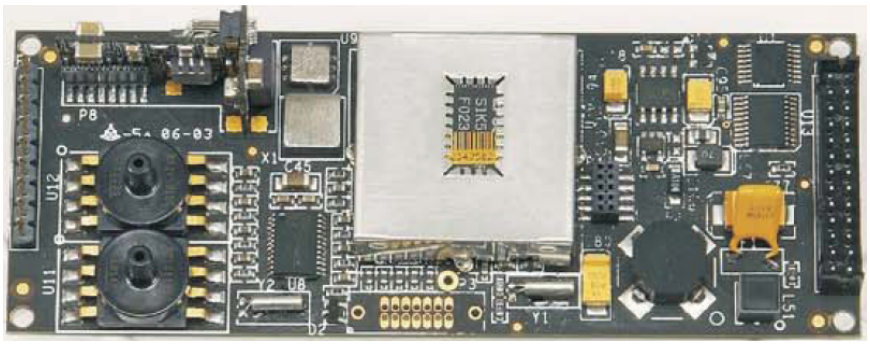

Figure 2.2: MP2028^{g} Autopilot Hardware [7]

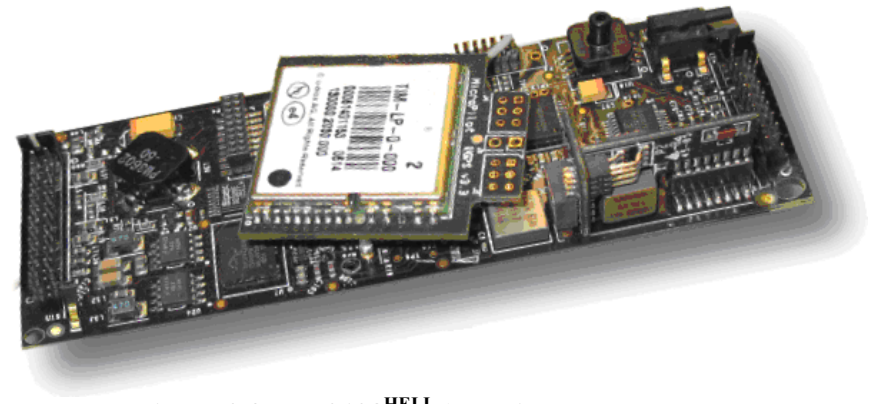

Figure 2.3: MP2128^{HELI} Autopilot Hardware [7]

The flight control algorithm runs at 30 Hz and is based on PID control loops with gain scheduling and feed-forward parameters. The autopilot software supports autonomous takeoff, landing and GPS waypoint navigation with altitude and airspeed hold. Hand launch, bungee launch and catapult launch are also supported. Control parameters, flight-paths and commands may be adjusted in real-time while the UAV is in flight. Custom user control loops, logging functions and telemetry fields may be added to the autopilot by using the separate XTENDER^{mp} package.

The MP2128^{HELI}, shown in Figure 2.3, has identical dimensions and weight as the MP2028^{g}. It utilizes a RISC microprocessor with approximately 50 times the performance and double the memory of the MP2028g. The sensor system has been improved with an integrated 4 Hz GPS, faster sampling rates, and a 12-state Kalman filter. An ultrasonic altimeter has also been added in order to support vertical takeoff and landing. The data logging and telemetry

update rates are adjustable from 5 to 30 Hz. The servo update rate is also selectable from 50 to 200 Hz. The autopilot software features all the capabilities of the MP2028^{g}, can also handle rotary-wing aircraft, and runs at a maximum update rate of 180 Hz [7].

2.2.2 Procerus Technologies: Kestrel Autopilot

The Kestrel is an autopilot system designed for use with small and very lightweight fixed-wing aircraft. It was originally developed through research at Brigham Young University's MAGICC laboratory [8] and is now available as a commercial product from Procerus Technologies [9].

Version 2.4 of the autopilot hardware, shown in Figure 2.4, weighs only 16.7 grams and measures 2.00" by 1.37" by 0.47", making it the smallest and lightest commercial autopilot system currently available. The system features a full inertial measurement unit (IMU) that includes 3-axis rate gyros, accelerometers, and a 3-axis magnetometer. Differential and absolute pressure sensors are also included for measuring airspeed and altitude. The sensor system uses three onboard temperature sensors in conjunction with a special compensation algorithm to reduce sensor drift. A GPS receiver is not included onboard. Instead, one of four RS-232 serial ports must be used to connect to an external GPS device. An optional "piggy-back" header can be used to connect a MaxStream, Microhard, Freewave or Aerocomm radio modem directly to the autopilot board. The system has built-in support for controlling four servos, but an external add-on board can increase this to a total of twelve servos. External hardware must also be used if manual control override from an RC transmitter is desired.

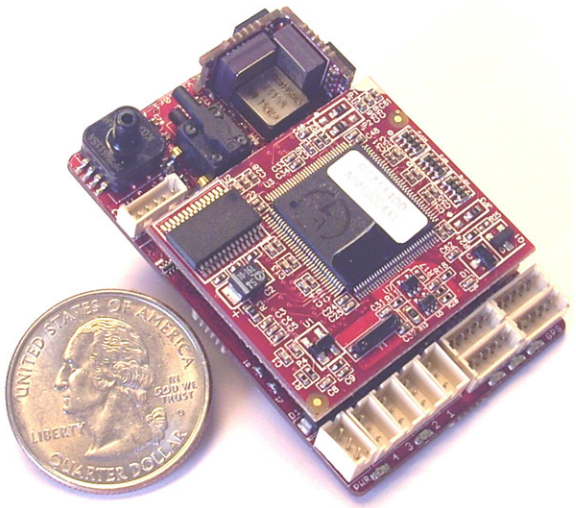

Figure 2.4: Kestrel 2.4 Autopilot Hardware [9]

An 8-bit, 29 MHz Rabbit 3000 microcontroller with 512KB of RAM and 512KB of Flash is used for the autopilot software. The flight control algorithm is PID-based and supports autonomous takeoff, landing, and waypoint navigation modes. Semi-autonomous altitude and airspeed modes are also available. Control parameters and navigation waypoints can be changed in real-time and partially automated PID tuning is possible. The autopilot can also be used with forward-looking, side-looking and gimbaled camera systems. Some user customization of the autopilot is possible using a separately licensed development kit from Procerus [9].

2.2.3 Cloud Cap Technology: Piccolo Series

Cloud Cap Technology currently manufactures two autopilot systems capable of flying both fixed-wing and rotary-wing UAVs. The Piccolo II autopilot has the highest number of I/O interfaces and is intended for advanced, payload heavy applications. The Piccolo SL is a newer and physically smaller autopilot system with reduced user I/O capabilities. Both systems, shown in Figure 2.5, share equivalent sensor, communication and processing hardware and run the same flight control software.

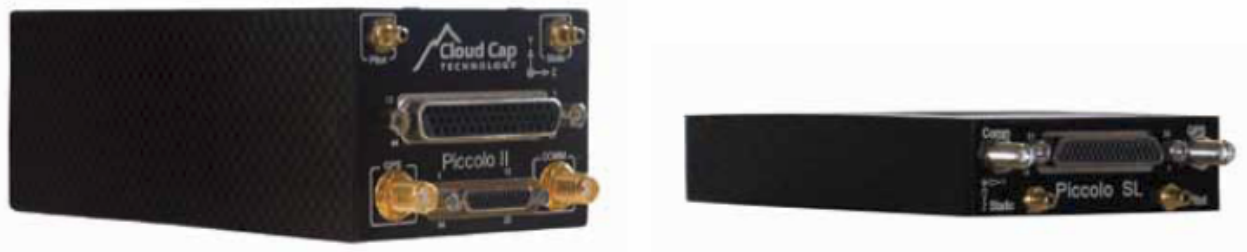

Figure 2.5: Piccolo II Autopilot (left) and Piccolo SL Autopilot (right) [10]

The Piccolo SL is a highly integrated autopilot measuring 5.16” by 2.19” by 0.76” and weighing 124 grams. The flight control processor is a 40 MHz Motorola MPC555 based microcontroller with an integrated floating point unit (FPU). The onboard sensor system includes a 4 Hz uBlox GPS receiver, 3-axis accelerometer, 3-axis gyroscope, static pressure sensor and pitot pressure sensor. External magnetometers and laser altimeters are supported as an option. The system also features an integrated RF data link with a range of available frequency options, including 900 MHz and 2.4 GHz.

Several autopilot software packages are offered and a source code license can be obtained, allowing for user customization. Different control algorithms are available, including classic PID-based controllers for both fixed-wing and rotary-wing and a neural-network-based helicopter controller developed by Guided Systems Technologies. Autonomous takeoff, landing, GPS waypoint navigation, and stability augmentation modes are possible. Control parameters and navigation waypoints can be changed while in-flight [10, 11].

2.2.4 Adaptive Flight: FCS20

The FCS20 is an integrated FCS designed to allow for advanced flight and mission capabilities in small UAVs. It was originally developed at the UAV Research Center at Georgia Institute of Technology [12]. It is now being sold commercially by Adaptive Flight Inc. [13]. The system has been successfully tested in several vehicle types, including fixed-wing, ducted fan and ultra-small helicopters.

Figure 2.6: FCS20 Autopilot Hardware [13]

The autopilot hardware, shown in Figure 2.6, measures 55mm by 85mm by 28mm and weighs 65 grams. The system consists of two PCBs: one containing the power supply, sensors and communications interface, and a second board that contains the processing system. The sensor board includes 3-axis rate gyros and accelerometers, absolute and differential pressure sensors and a 4 Hz uBlox GPS module. Four RS-232 serial ports, 80 general purpose I/O pins, and an Ethernet interface are provided. An external 5.17 megapixel CMOS sensor is also available.

The processor board includes a 300MHz floating point DSP from Texas Instruments and a high-end Altera Stratix II FPGA. The FPGA interfaces with the onboard sensors, external communications hardware, and servos. It also preprocesses the sensor data and communicates with the DSP via a FIFO-based bus interface. The DSP is responsible for running the autopilot software, which can implement some combination of flight control and image processing. Exact details on the autopilot software are not available for the commercial system. However, an extended Kalman filter and a neural network based flight controller were tested during development at Georgia Tech [12, 13].

2.2.5 Rotomotion: AFCS

The Automatic Flight Control System (AFCS) from Rotomotion is an integrated autopilot designed for use with RC helicopters. It combines the flight control processor with an attitude and heading reference system (AHRS), GPS receiver, servo controller and an optional 802.11 wireless link in an aluminum enclosure weighing 1.25 pounds. The hardware is shown in Figure 2.7, below.

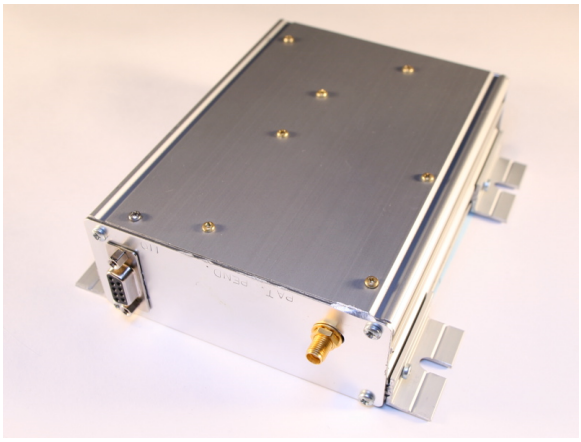

Figure 2.7: Rotomotion AFCS Hardware [14]

The AHRS includes a magnetometer, accelerometers, and gyroscopes, but does not include a barometric pressure sensor for altitude measurements. Altitude information is instead obtained from the GPS receiver. The servo controller includes a safety system that allows for manual control of the helicopter servos using a standard RC transmitter. The flight control software is PID based and runs on a 206MHz Intel StrongARM SA-1110 CPU. It allows for autonomous waypoint navigation, a fast forward flight mode, and a semi-autonomous velocity command mode. Takeoff and landing must be performed manually however. The autopilot source code is available to subscribers which allows for examination and modification [14]. It was originally based on the open-source autopilot project at [15].

2.2.6 weControl: wePilot1000

The wePilot1000 is an autopilot system developed by weControl for use small unmanned helicopters. The flight controller is based on H-infinity methods and provides robust system performance. It does, however, require recorded data from a manually piloted flight before the system can be adapted to a specific helicopter airframe. The flight control software allows for fully autonomous takeoff and landing, waypoint navigation and semi-autonomous velocity command modes. Flight paths and mission parameters can be changed during flight.

Figure 2.8: wePilot1000 Autopilot Hardware [16]

The system, shown in Figure 2.8, weighs 1020 grams, measures 120mm by 154mm by 125mm and features a built-in vibration isolation system. The embedded computer system features an Intel XScale PXA255 Processor with 32MB of Flash and 64MB of SDRAM along with a Xilinx SpartanXL FPGA. The sensor system consists of a Novatel GPS receiver, 3 accelerometers, 3 rate gyros, a barometer and an external magnetometer. An extended Kalman filter is used for data fusion of the GPS and inertial sensors. Five RS-232 serial ports, 6 analog inputs and 8 digital I/O channels are also available for interfacing with external sensors, payloads and data-links. Ten PWM input channels are provided for connecting to the RC receiver, along

with ten PWM outputs for aircraft servos. It is possible to completely bypass the system, which allows for direct manual control of the aircraft by the pilot [16].

2.3 Research Level Autopilots

2.3.1 Institute for Scientific Research, Inc.

The Institute for Scientific Research has created the Adaptive Autopilot Microcontroller (AAM), a hardware platform designed to be scalable to a wide range of UAV applications and compatible with small fixed-wing airframes. This system implements a dual-processor architecture consisting of a "flight processing unit" and a "sensor data acquisition unit". Each processing system has its own PCB and they are interfaced through a single common header. The large amount of overhead involved in sampling and filtering the sensor data was cited as the motivation for utilizing two separate processors. This improved the timing determinism of the flight processor which would have otherwise been degraded by external sensor interrupts [17].

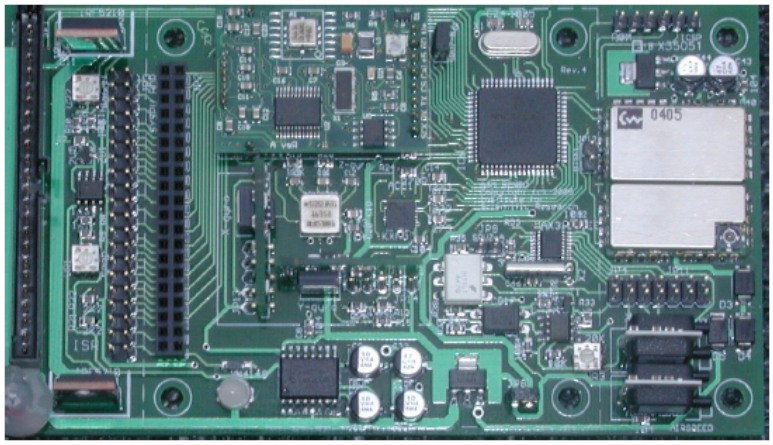

Figure 2.9: Sensor Data Acquisition Unit for AAM [17]

The sensor data acquisition board (SDAQ), shown in Figure 2.9 above, employs a 30MHz PIC microcontroller that runs at an update rate of 200Hz and performs digital filtering of the analog sensor data. The board integrates gyroscopes, accelerometers, GPS receiver, tilt compensated compass, barometric pressure sensors, temperature sensor and battery monitor. It

also generates the PWM signals necessary to control the aircraft's servo actuators. Sensor data is aggregated in the SDAQ's internal buffer and is later polled by the flight processor at the start of its update cycle.

The flight processor board utilizes a 32-bit 100MHz Etrax 100 system on chip (SoC) processor and runs the Linux 2.6 operating system. The processor interfaces with a 900MHz radio modem for ground station communications and makes use of a 128MB USB flash drive for data logging. MATLAB/Simulink was used to auto generate C code for the autopilot software. This software implements an extended Kalman filter and an adaptive PID controller that runs at a 25 Hz update rate [17].

2.3.2 University of South Florida

An autopilot platform was recently developed at the University of South Florida as the result of graduate research [18]. The goal of the research was to design a flexible hardware platform that allowed for hardware-in-the-loop simulation and rapid prototyping of new control theory. The resulting system, shown below in Figure 2.10, utilized a Xilinx Spartan 3-1400AN FPGA as its main processing unit. The MATLAB/Simulink System Generator was used to create processing, control, and sensor sub-system blocks that were implemented in the FPGA fabric.

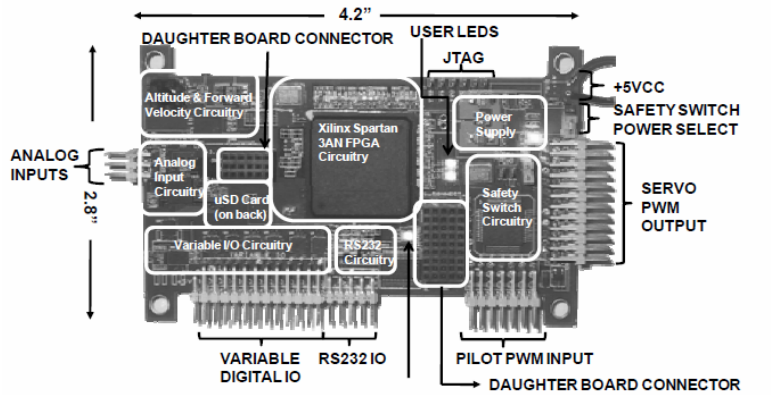

Figure 2.10: USF Autopilot Hardware [18]

Barometric pressure sensors were included onboard for measuring altitude and forward airspeed. An external Microstrain IMU and SuperStar II GPS were used for attitude and position information. The board provides 24 variable voltage digital inputs, 3 analog inputs, and 4 RS-232 serial ports to allow for external sensors and communications hardware to be added. An optional, application specific daughter board can also be interfaced to the main board. The board also integrates a safety-switch circuit that allows for manual override of the servos in the event of a system failure. This hardware platform has been successfully used as a PID-based control system for an RC truck robot [18, 19].

2.3.3 University of Singapore

The University of Singapore has developed a flight control system for small UAV helicopters, known as the HeLion. The system utilizes a PC/104 embedded computer stack, shown in Figure 2.11 below, containing a main processor board, a serial communications board, an analog data acquisition board, a servo controller board, and a DC power supply board. The processing board includes an Intel Celeron 650 MHz processor, 512MB of SDRAM, 2 GB of compact flash memory and two RS-232 serial ports. The serial communications board adds an additional 8 serial ports and the data acquisition board adds 16 analog input channels. These boards are used to interface with an external Crossbow NAV420 IMU/GPS module, a UPK500 ultrasonic range finder, a Freewave 2.4 GHz radio modem, and RPM and fuel level monitoring equipment.

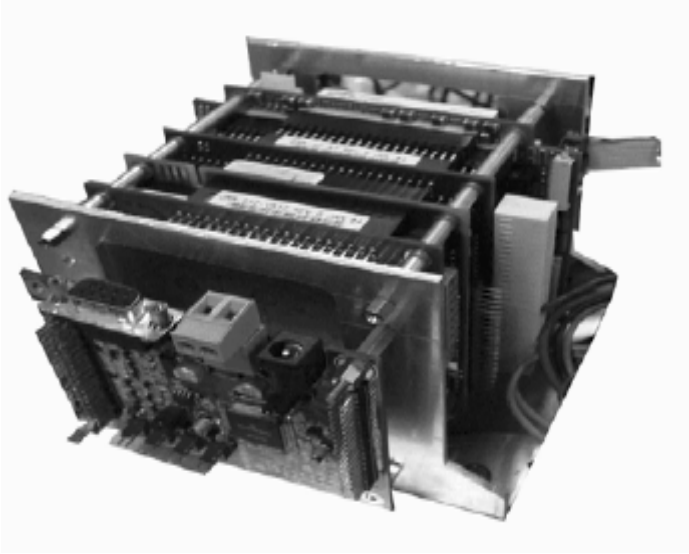

Figure 2.11: HeLion Autopilot Hardware [20]

The QNX Neutrino real-time operating system (RTOS) runs on the Intel CPU and executes the multi-threaded autopilot software. The autopilot system implements a behavior-based, task scheduling architecture that executes a series of basic operations such as hovering, forward flight, commanded heading, speed control and trajectory tracking. These basic behaviors are used to complete more complex tasks and flight patterns. The actual flight control algorithms use a custom composite nonlinear feedback (CNF) controller [20, 21].

2.3.4 University of Colorado

A distributed avionics system for use in small UAVs has been designed by the University of Colorado. This system utilizes a modular architecture consisting of several independent, interconnected processor nodes. Each node is responsible for a particular function, such as connecting to a set of sensors or handling a computational task. The primary benefit of this approach is said to be the increased flexibility afforded, allowing sub-systems to be easily added or removed. Other stated advantages include the ability to place individual nodes close to the sensors they are monitoring, which can reduce susceptibility to noise, and the possibility of increased fault tolerance when used in a redundant configuration [22].

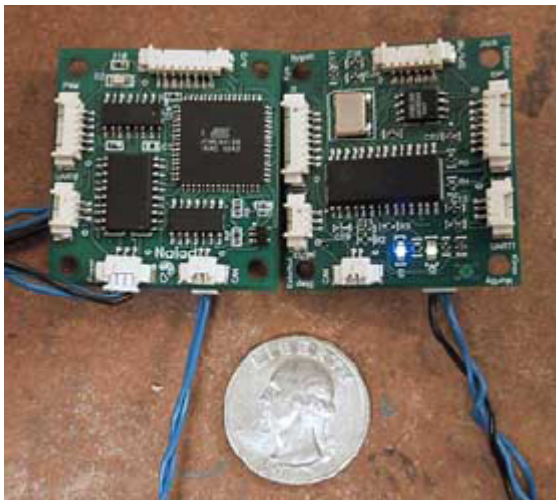

Figure 2.12: Two Naiad Nodes [22]

Each node, known as a "Naiad", is approximately 1.5 inches by 1.5 inches in size and contains a 14 MHz ATmega128 microcontroller. Two such nodes are shown above, in Figure 2.12. The 8-bit microcontroller on each node provides several I/O interfaces, such as: PWM outputs, TTL and RS-232 UARTs, and analog inputs. All nodes communicate using the controller area network (CAN) bus protocol and use a message based addressing scheme.

The first application of the system has been in the creation of a stability augmentation system for a small fixed-wing UAV. Nodes were used to interface with IMU and GPS modules, communicate to a ground station over a 900 MHz wireless link, perform stability calculations, mix user input and control input, and output PWM signals to the servos [22].

2.3.5 Utah State University

The AggieAir project at Utah State University is a flight control and payload management system intended for small, multi-UAV applications. It consists of the following interconnected systems: AggiePilot, AggieNav and AggieCap [23]. AggiePilot is a modified version of the open source Paparazzi autopilot system developed at ENAC University in France [24]. It has been adapted to add support for the Joint Architecture for Unmanned Systems (JAUS), a command

and control architecture designed by the US Department of Defense to support interoperability of unmanned systems.

Figure 2.13: AggieNav Hardware with Gumstix and GPS [23]

AggieNav is a custom inertial navigation system (INS) that interfaces with AggiePilot via an SPI bus. The AggieNav hardware, shown in Figure 2.13, is based on the 32-bit AVR32A256B microcontroller and runs at 60 MHz. It integrates the Analog Devices ADIS1635x IMU, the Honeywell HMC6343 tilt-compensated magnetic compass, the uBlox LEA-5H 4Hz GPS and two barometric pressure sensors from VTI. The AggieNav software samples the raw sensor data at 100 Hz and sends it via a TTL serial port to an attached Gumstix processor module. The Gumstix, a 600 MHz XScale based module running Linux, executes an extended Kalman filter process on the raw sensor data. The filtered data is then passed to AggiePilot via the AggieNav. The Gumstix processor module is also responsible for running the AggieCap software, which handles payload control and mission planning. An optional 2.4GHz WiFi link can be attached to the Gumstix allowing the transfer of high resolution images to the ground station [23, 25].

2.3.6 Istanbul Technical University

Istanbul Technical University has created a multi-bus, interconnected micro-avionics system designed for use with mini-helicopters, fixed-wing aircraft and ground vehicles. This system consists of numerous processor, sensor, and communications boards that are connected together via a combination of Ethernet and CAN bus backbones.

The main processing unit is a Motorola MPC555 PowerPC processor running at 40 MHz. It has an FPU, 26kB of RAM, 448kB of Flash and a modular I/O system that allows it to read and write PWM control signals. This processor executes autopilot code that is generated using the MATLAB/Simulink Real-Time Workshop. An LPC2294 ARM7-based microcontroller running at 60 MHz is used to interface with an XTend 900 MHz radio modem and communicate with the ground station. This processor board is also used for digital filtering of the sensor data. A Tiny886ULP PC104+ stack is utilized for flight management and multi vehicle coordination processing. This board utilizes a 1GHz Crusoe processor with an FPU and 512MB of SDRAM. Finally, a Gumstix Connex single board computer is used for information distribution and mission planning. This board uses a 400 MHz XScale PXA255 processor with 64MB of RAM and runs the Linux 2.6 operating system. These four processor boards are shown below in Figure 2.14.

Figure 2.14: The Processor Portion of the ITU Avionics Package [26]: LPC2294 ARM Board (1), MPC555 Board (2), Tiny886ULP PC104+ (3), and Gumstix with Robostix and netCF (4)

The sensor system consists of several discrete devices for measuring position and orientation. This includes a Garmin 15H GPS receiver, a Crista IMU from Cloud Cap Technology, a Honeywell HPA200W5DB barometric altimeter, and a Honeywell HMR3300 magnetometer. A Daventech SRF10 ultrasonic rangefinder and an Opti-Logic laser rangefinder are also used during takeoff and landing. Custom SmartCAN boards are used to interface these sensors with the inter-processor CAN bus. A custom safety-switch board is also used to allow switching between autonomous and manual control of the aircraft servos. These custom boards make use of 8-bit PIC microcontrollers from Microchip Technology.

2.4 Hardware Comparison

A direct comparison of the previously covered autopilot hardware in terms of computational capabilities is shown below, in Table 2.1. The following information will be provided for each system's primary CPU: architecture/model (family), word-size (word), clock frequency (speed), floating point capability (FPU) and amount of main memory (RAM). If a system features programmable logic, the type of FPGA will also be listed. In some cases, not all information is available due to a system's proprietary nature.

Table 2.1: Comparison of Autopilot Hardware

Organization	System	Central Processing Unit					Programmable Logic
		Family	Word	Speed	FPU	RAM	
MicroPilot	MP2028g	Motorola 68332	32-bit	?	No	1.5 MB	No
MicroPilot	MP2128heli	Unknown, RISC	32-bit	?	?	3 MB	No
Procerus Technologies	Kestrel 2.4	Rabbit 3000	8-bit	29 MHz	No	512 KB	No
Cloud Cap Technology	Piccolo SL	Motorola MPC555	32-bit	40 MHz	Yes	?	No
Adaptive Flight	FCS20	Texas Instruments DSP	32-bit	300 MHz	Yes	256 MB	Yes, Altera Stratix II
Rotomotion	AFCS	Intel StrongARM SA-1110	32-bit	206 MHz	No	?	No
weControl	wePilot1000	Intel Xscale PXA255	32-bit	?	No	64 MB	Yes, Xilinx Spartan-XL
Institute for Scientific Research, Inc.	AAM (FCS)	Axis Etrax 100	32-bit	100 MHz	No	16 MB	No
University of South Florida	Autopilot	None, VHDL Only					Yes, Xilinx Spartan-3AN
University of Singapore	HeLion	Intel Celeron	32-bit	650 MHz	Yes	512 MB	No
University of Colorado	Naiad	Atmel ATmega128	8-bit	14 MHz	No	4 KB	No
Utah State University	AggieNav	Atmel AVR32	32-bit	60 MHz	No	256 KB	No
Istanbul Technical University	Main Board	Motorola MPC555	32-bit	40 MHz	Yes	26 KB	No
Virginia Commonwealth University	Suzaku-FCS	Xilinx MicroBlaze	32-bit	48 MHz	Yes	16 MB	Yes, Xilinx Spartan-3

Chapter 3: System Hardware

This chapter will present a detailed discussion of the physical hardware developed for the new autopilot platform, along with the relevant design constraints and considerations. This will include information on the specific FPGA-based processor modules and the custom multi-board architecture utilized in this design. Implementation details will be covered, including PCB layout, power regulation, signal conditioning, and the various control and measurement interfaces. Information on the specific sensors and communications hardware used will also be provided. Analysis of potential system failure modes and any built-in fail-safe measures will be discussed where applicable.

3.1 System Requirements & Considerations

One of the primary requirements of the new system is significantly improved processing capability, sufficient to handle the computational demands of more complex control algorithms. Ideally, CPU performance should be at least 50 to 100 times greater than that of the previous generation hardware. As a rough benchmark, the main system processor should be fast enough to run the current control and navigation algorithms at 200 Hz, while using less than 25% of the available CPU time. The system must also achieve hard real-time performance, with low timing jitter and low control loop latency. A second processor or microcontroller is highly desirable, and should be used to offload low-level jobs, such as sensor interfacing and filtering, communications, and other I/O heavy tasks. At least one of the two processor devices should contain programmable logic in order to maintain the flexibility of the previous system.

The system must also maintain compatibility with all currently used sensors, while also allowing the use of additional types of sensor devices. All analog sensors must be properly filtered and achieve high immunity to both radiated and conducted EMI. This includes both onboard noise sources such as digital circuits and switching regulators, as well as external noise sources such as electric motors. The system must support at least 8 analog input channels and achieve a final sample resolution of at least 16 bits. The analog-to-digital converter must be capable of continuously sampling all analog channels at rates greater than 5 kHz.

The system must support the use of a high throughput wireless link, capable of supporting telemetry rates of up to 20 Hz. The wireless link must operate in a frequency band other then 2.4 GHz and must not interfere with the RC safety link. The wireless link should support both point-to-point and point-to-multipoint modes of operation. This will allow it to maintain backwards compatibility with the current ground station software, while enabling support for future multi-UAV collaborative applications. The system must also include the option of at least 2GB of removable non-volatile storage, suitable for mission logging purposes.

Safety switch hardware must be included onboard, and support switching of up to 8 servo control channels. This hardware is safety-critical; the switch must remain functional and allow for manual override as long as the aircraft servos are powered. Loss of FCS power must not impede the safety switch's operation. Power protection circuitry should also be added to increase system resilience to transient voltage faults and possible user error. Above all, the system must maintain compatibility with all previous airframes and should support the use of the current enclosures. Maximum dimensions of 4" x 3" x 3" were imposed to ensure these requirements were met.

3.2 Hardware Overview

The architecture of the new autopilot platform utilizes a stacked multi-board design, consisting of a primary "Main Board", a secondary "Auxiliary Board", and two FPGA-based processing modules. The overall board arrangement is shown below, in Figure 3.1.

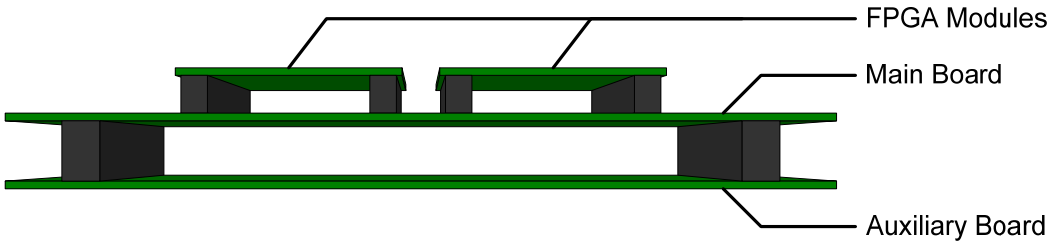

Figure 3.1: Board-Level Diagram of System Architecture

The two FPGA modules are denoted the "Flight Control Module", or FCM, and the "Instrumentation Control Module", or ICM, and all processing tasks are divided among them. The FCM is responsible for handling all high-level autopilot functionality, including the flight control algorithms and navigation routines. The ICM is responsible for lower-level tasks, such as sensor interfacing, digital filtering, actuator control, and wireless communications. The ICM queues up all sensor data and ground station commands, sending them to the FCM at a rate of 200 Hz. The FCM then processes the received data, executes the control algorithms, and responds with the necessary commands and telemetry data. All inter-module communications occur via a low-latency, full-duplex bus and utilize a simple, generic protocol. Additional details of the two FPGA modules will be provided in the following section.

This dual core configuration provides several key benefits. The primary advantage is improved system throughput and timing consistency due to the use of parallel processing. The ICM handles all of the I/O heavy tasks which tend to incur a high number of processor interrupts. This allows the FCS algorithms to execute with minimal interruptions, which in turn helps to reduce performance degradation from effects such as context switching overhead and cache

thrashing. This division of labor can also potentially aid with development of the autopilot software. It allows the FCS developer to focus on improving the control algorithms without worrying about the lower-level details of the sensor system. It also allows for changes and additions to be made to the sensor configuration without affecting the FCS software.

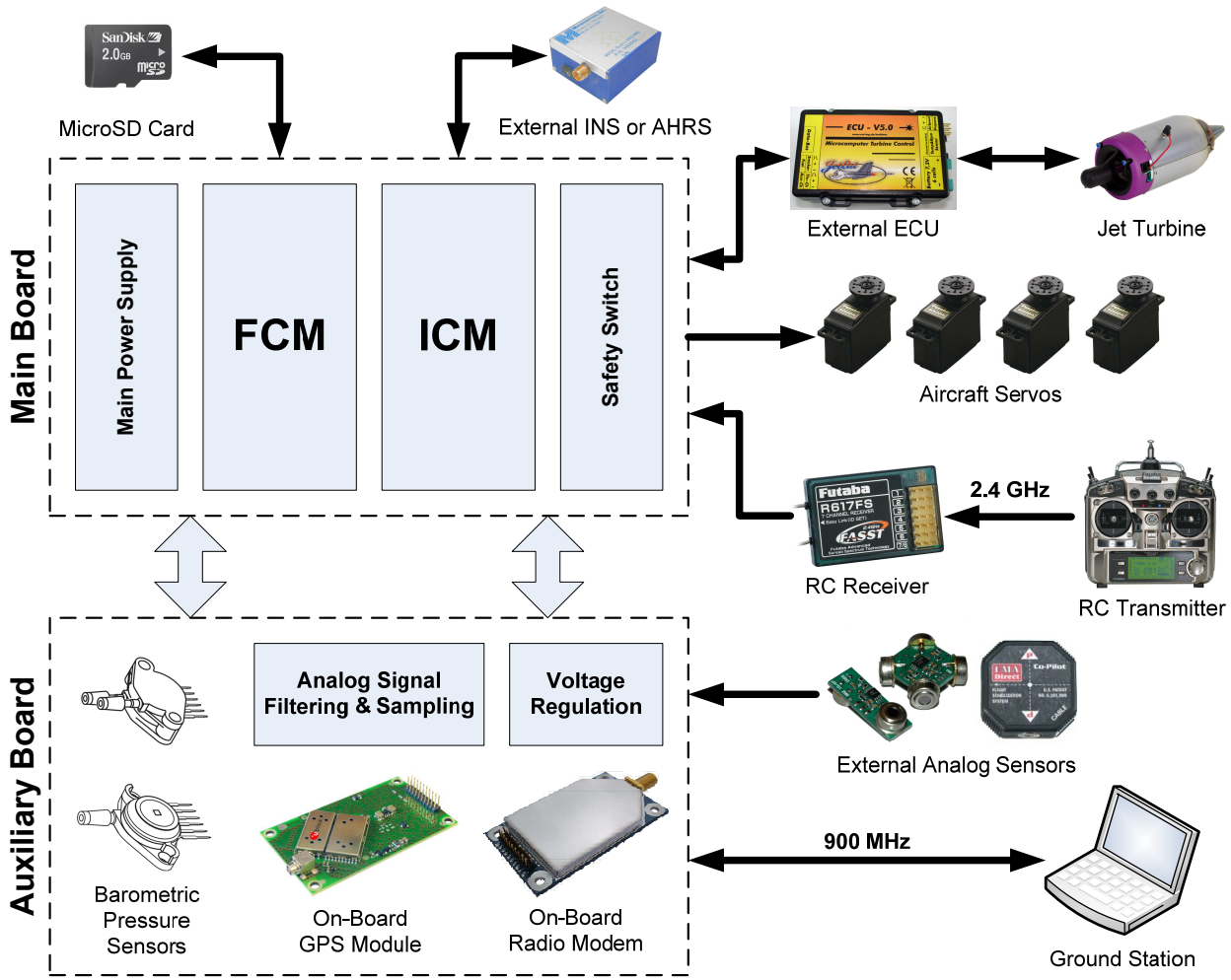

Figure 3.2: Component-Level Diagram of System Architecture

As shown above in Figure 3.2, the system is also divided between a main board and an auxiliary board. The main board interfaces with the two FPGA modules, and contains support circuitry needed by any autopilot implemented on this system. This includes power regulation and monitoring, digital interfaces such as RS-232 and SPI, and an onboard actuator safety switch, among other things. The auxiliary board is intended to be a low-cost add-on board that

can be customized to include application specific components and devices. In this case it includes analog signal conditioning hardware, barometric pressure sensors, a GPS receiver, and a radio modem. This arrangement improves system flexibility, while also providing an additional layer of isolation between the sensitive analog components and the noisy digital circuitry. Several external sensors and devices can also be connected to the two boards, as shown in the above figure. Additional details of the main board and auxiliary board will be given in sections 3.4 and 3.5, respectively.

The fully assembled autopilot hardware is shown below in Figure 3.3 and Figure 3.4. Overall dimensions of the complete system are 3.8" x 3.0" x 2.75". Length and width remain identical to that of the previous system, while total height is only increased by ¼ inch.

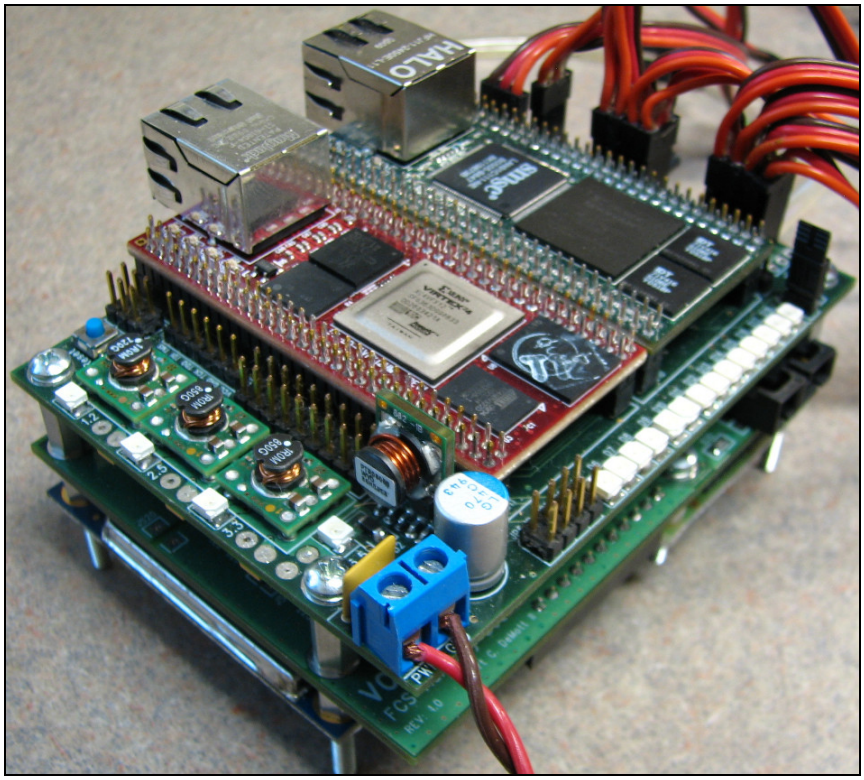

Figure 3.3: Fully Assembled Autopilot Hardware, Angled View

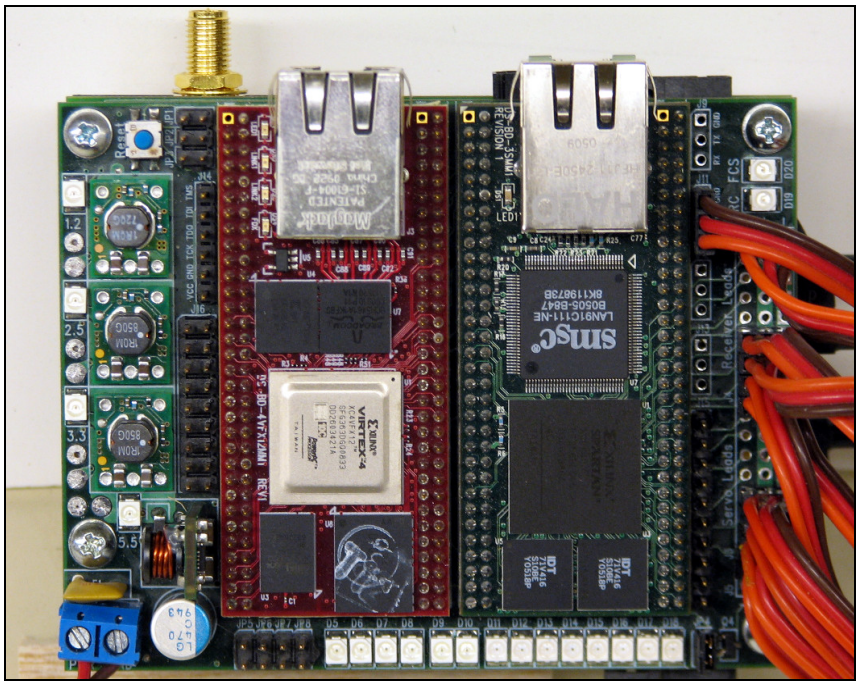

Figure 3.4: Fully Assembled Autopilot Hardware, Top View

3.3 FPGA Mini-Modules

Processing power and programmable logic are provided by two FPGA-based System-on-Module (SOM) devices developed by Avnet, Inc. These are the Xilinx Virtex-4 FX12 Mini-Module and the Xilinx Spartan-3 Mini-Module. Both modules are pin-compatible and have identical physical dimensions, measuring only 30 mm by 65.5 mm. Each module contains an FPGA, Flash memory, DDR or SRAM memory, an Ethernet interface, and 76 user I/O pins. Excellent documentation is available including detailed user manuals, full schematics, bills of materials, and several example projects including VHDL and C source code. Both modules make use of standard Xilinx development and programming tools [27, 28].

The Spartan-3 Mini-Module, as its name implies, is based around a Xilinx Spartan-3 FPGA. The Spartan-3 XC3S400 FPGA provides the equivalent of 400 thousand system gates and allows the use of the MicroBlaze soft-core processor. This processor core is highly configurable and can operate at speeds up to 75 MHz. The module includes 2MB of Flash

memory, 1MB of high-speed single cycle SRAM, and a 100Mbps Ethernet interface. By itself, the Spartan-3 Mini-Module provides near feature parity and increased performance compared to the Suzaku-S used in the previous generation autopilot. A picture of the Spartan-3 Mini-Module is shown below in Figure 3.5 and a high-level block diagram is provided in Figure 3.6.

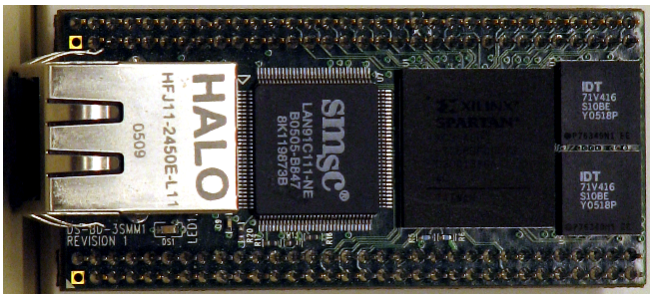

Figure 3.5: Avnet Spartan-3 Mini-Module

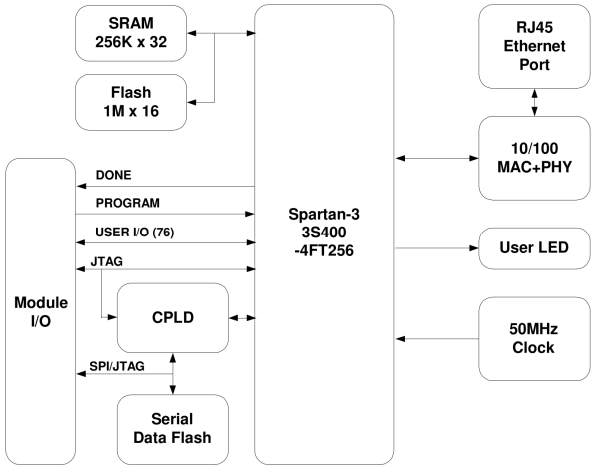

Figure 3.6: Avnet Spartan-3 Mini-Module Block Diagram [28]

The FX12 Mini-Module provides both substantially higher performance and additional functionality compared to the Spartan-3 version. The onboard Virtex-4 FPGA is capable of operating at significantly higher clock frequencies and contains over 50% more programmable logic resources than the Spartan-3. It also includes several hard IP blocks, including a PowerPC 405 processor, a Tri-Mode Ethernet MAC, and several dedicated DSP slices. In addition to the Virtex-4 FPGA, the FX12 Mini-Module includes 64MB of DDR SDRAM, 4MB of Flash

memory, and a Gigabit Ethernet interface. A picture of the Mini-Module is shown below in Figure 3.7 and a block diagram is provided in Figure 3.8.

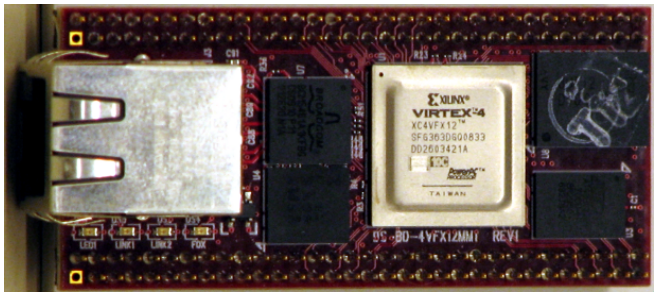

Figure 3.7: Avnet FX12 Mini-Module

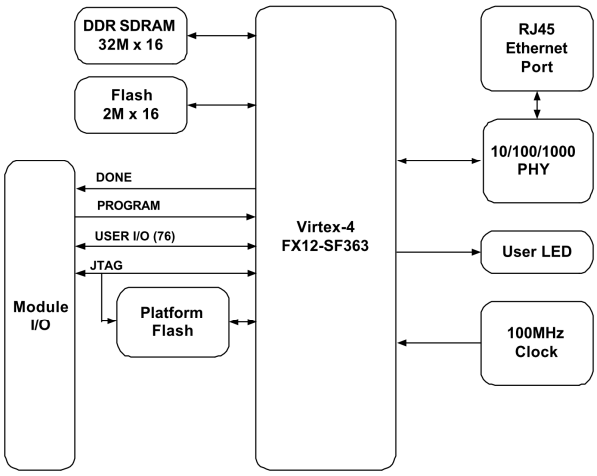

Figure 3.8: Avnet FX12 Mini-Module Block Diagram [27]

The FX12's built-in PowerPC CPU can operate at frequencies in excess of 300 MHz and achieves a performance efficiency of approximately 1.5 MIPS/MHz. Auxiliary processing units (APUs) can be implemented in the FPGA to augment the PowerPC, allowing for acceleration of computationally intensive operations and algorithms. The embedded PowerPC processor allows the use of several possible operating systems, including Linux-based systems with real-time extensions. This allows standard open-source development and debugging tools to be used, easing development and testing of the high-level autopilot software without sacrificing real-time performance.

The small size of these two mini-modules allows for the desired dual processor architecture without increasing main board dimensions compared to the previous generation hardware. This helps maintain a small overall system footprint, preserving airframe compatibility. Increased flexibility is afforded thanks to the programmable hardware in the two FPGAs. The use of familiar Xilinx tools also simplifies development by leveraging the knowledge obtained from previous FPGA-based systems.

A high degree of modularity is achieved due to the pin-compatible nature of the two modules. The system presented in this paper utilizes an FX12 Mini-Module for the flight control and navigation routines (FCM), while a Spartan-3 Mini-Module is used for the sensor monitoring, wireless communications, and actuator control tasks (ICM). Several other configurations are also possible, however. In cases where additional processing power is required, the Spartan-3 Mini-Module could be replaced with a second FX12 Mini-Module. This may be beneficial if advanced flight control and navigation algorithms must be combined with other complex tasks, such as multiple vehicle collaboration and target tracking applications. Single module configurations are also a potential possibility when computational requirements are significantly lower or timing requirements are less strict.

Other processors and FPGA devices were also evaluated, but ultimately rejected for several reasons. Initially, the use of a single Virtex-4 FX20 FPGA was considered in lieu of the two Mini-Modules. This FPGA has substantially higher logic capacity than both Mini-Module FPGAs, giving it the ability to house a soft MicroBlaze core in addition to the embedded hard PowerPC core. This would satisfy the dual processor requirement while saving board space. This idea was quickly abandoned, however, due to cost and fabrication concerns. By itself, this FPGA would cost slightly more than both Mini-Modules together, not including the additional

supporting components needed. It is also only available in BGA packages and requires a substantial amount of decoupling capacitors, many of which must use case sizes of 0402, to adequately bypass high frequencies. These requirements would significantly increase board fabrication cost and prevent in house assembly.

The possibility of using an FX12 Mini-Module along with a dedicated onboard Spartan-3/3E device was also considered in order to save cost. The lower-end Spartan-3/3E devices from Xilinx are available in non-BGA packages and have lower bypassing requirements, making this option potentially feasible. However, when combined with the other necessary components including Flash memory, SDRAM, and the Xilinx configuration IC, the cost savings decreased significantly. Board area would also noticeably increase, violating the size requirements imposed on the system. Other Spartan-3 based SOM devices, such as the TE0300 Industrial MicroModules from Trenz Electronic, were also considered. Although some were slightly cheaper than the Avnet equivalent, they too would require additional board real-estate.

Another notable device considered was the phyCORE-MPC5200-I/O from PHYTEC. This module includes a 32-bit Freescale MPC5200B processor operating at 400 MHz, along with an Altera Cyclone II FPGA. It also contains up to 128MB of SDRAM, up to 64MB of Flash memory, and a 100Mbps Ethernet PHY (no RJ-45 jack). This device provides substantial processing resources and natively supports real-time Linux, making it a good candidate for this project. Unfortunately, its board dimensions were nearly as large as the maximum permitted main board size and it provided insufficient clearance to place components directly beneath it. This would prevent the inclusion of necessary support circuitry without violating size constraints. Also, due to unfamiliarity with Altera devices, it could not be guaranteed that the Cyclone II FPGA had sufficient logic resources for this design.

3.4 Main Board

As mentioned previously, the main board contains the primary components necessary for the autopilot platform. This includes the main power supply and associated circuitry, safety switch hardware and servo connectors, non-volatile storage, RS-232 serial and digital I/O interfaces, and supporting components such as buffers and level-shifters. Also included are bulk decoupling capacitors, JTAG debugging interfaces, and status LEDs for the two FPGA modules.

3.4.1 PCB Design

The main board PCB is a six layer design measuring 3.8 inches by 2.9 inches. Dedicated ground and power planes are used to offer increased noise immunity and minimize voltage drop due to parasitic impedance. All high speed signals are routed on internal layers to reduce radiated EMI and limit susceptibility to external interference. Unrouted areas of internal signal planes are flooded with copper and grounded, further improving EMI shielding. Trace lengths are also kept as short as possible to prevent signal reflection issues. All bypass capacitors are placed close to the corresponding IC power pins and via-in-pad techniques are used when possible. This minimizes parasitic inductance, increasing decoupling performance at higher frequencies [29]. The actual layer stack-up and board layout are shown in Table 3.1 and Figure 3.9 respectively.

Table 3.1: Main Board Layer Stack-Up

#	Layer Type	Primary Functions
1	Top Layer	Low-Speed Signals, Components & Connectors
2	Inner Layer	Dedicated Ground Plane
3	Inner Layer	High-Speed Signals, +2.5V Power
4	Inner Layer	High-Speed Signals, +1.2V Power
5	Inner Layer	Dedicated +3.3V Power Plane
6	Bottom Layer	Low-Speed Signals, Components & Connectors

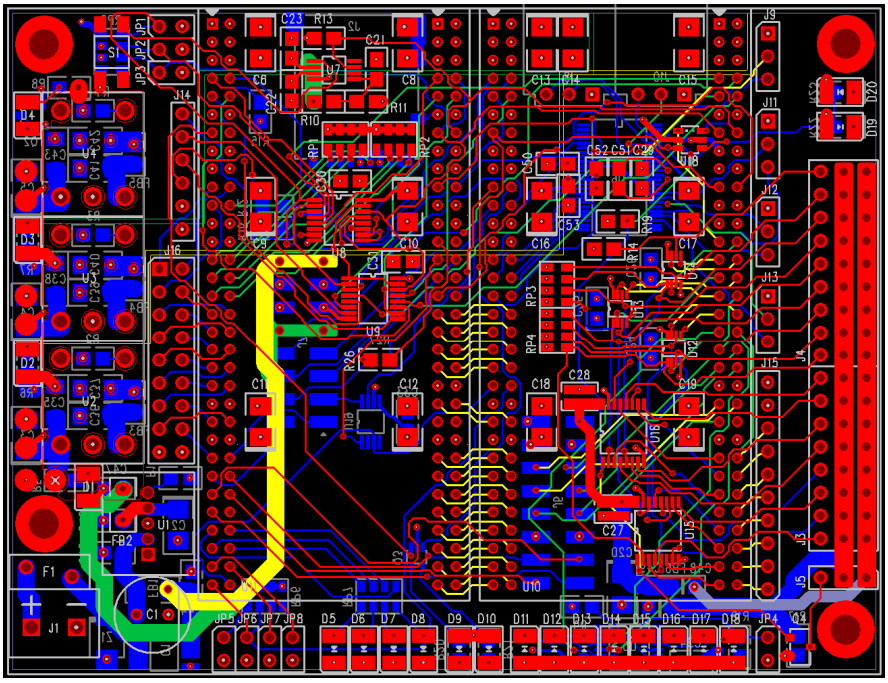

Figure 3.9: Main Board PCB Layout

3.4.2 Power Supply

A simplified diagram of the main board power supply is shown in Figure 3.10 below. It consists of three main stages: input protection, voltage regulation, and voltage monitoring. The input protection stage serves to prevent power faults from damaging the system. The voltage regulation stage provides the supply voltages needed by the FPGA modules and I/O interfaces on the main board, as well as providing power to the auxiliary board. The voltage monitoring stage ensures that the FPGAs are only active when the supply voltages are stable and within tolerance. These three stages will be covered in the following subsections.

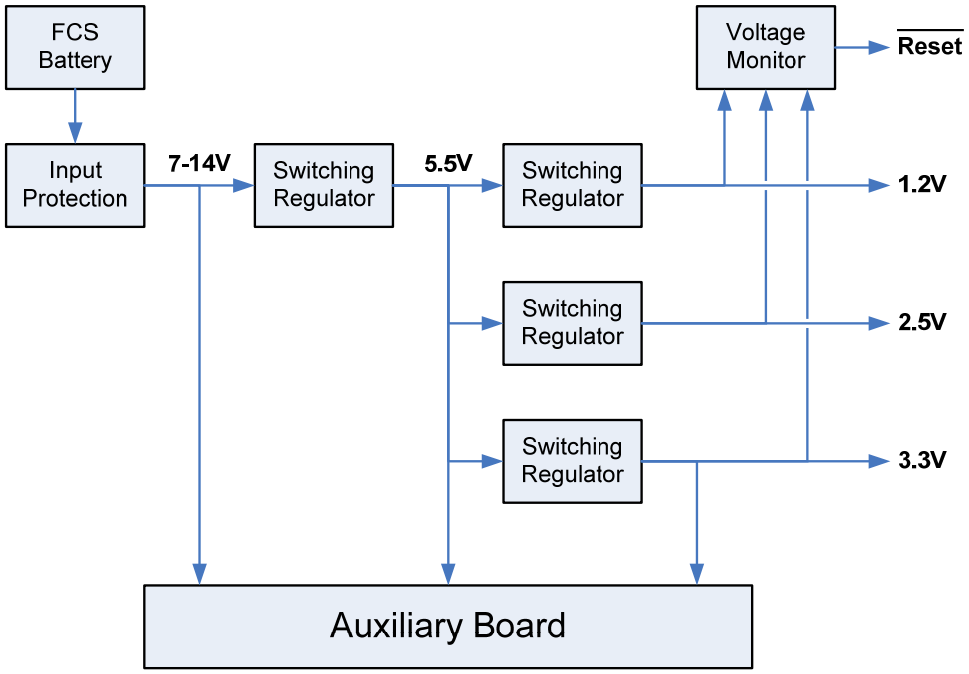

Figure 3.10: Main Board Power Supply

3.4.2.1 Input Protection

The input protection stage provides the system with transient voltage suppression (TVS), current limiting and reverse voltage protection. These features help prevent damage to the system by blocking or limiting the negative effects of external power disturbances. The actual protection circuit is shown below in Figure 3.11. It consists of a TVS diode (D1), a resettable fuse (F1), and a power MOSFET (Q1).

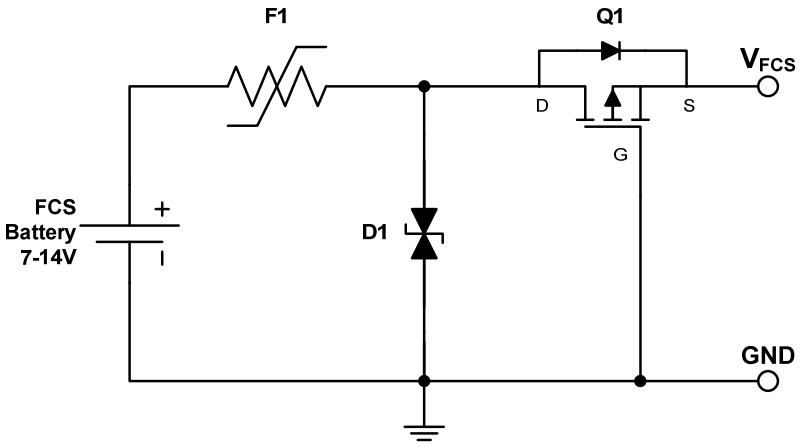

Figure 3.11: Input Protection Circuit

The TVS diode (D1) used is a bidirectional avalanche diode with a breakdown voltage of 14.4V. Under normal operating conditions, the diode will exhibit negligible leakage current on the order of a few microamps. However, if the input voltage exceeds ~14.4V, the diode will begin conducting due to avalanche breakdown and shunt the transient current to ground. This will effectively protect the system from momentary voltage spikes and surges by clamping the voltage that appears at the input [30]. Such disturbances can result due to load transients on the main battery or power supply. This includes possible overvoltage surges at power-on due to the combination of parasitic inductance in the power leads and the large low-ESR capacitance at the input of the voltage regulation stage [31].

The resettable fuse (F1) is a polymeric positive temperature coefficient (PPTC) device, commonly referred to as a "PolySwitch". It is essentially a temperature controlled resistor with a non-linear response. During normal operation the PPTC will exhibit a low on-resistance on the order of 0.01 to 0.1 ohms. If the current through the PPTC exceeds the trip current, the device will heat up and increase in resistance several orders of magnitude, thus limiting the current. It will remain in this state until the fault is removed [32]. When combined with the TVS diode, it can potentially protect the system from a prolonged overvoltage fault. It will also prevent the drawing of excessive current from the battery in the event that the system fails in a low-impedance state.

The power MOSFET (Q1) is a p-channel MOSFET with a low on-resistance of only 0.05 ohms. As shown in Figure 3.11, the gate is connected to ground, the drain is connected to the FCS battery (via the PolySwitch), and the source is connected to the input of the voltage regulation stage. This orients the intrinsic body diode in the direction of normal current flow. When the battery is installed correctly, a voltage of one diode drop below the battery voltage will

initially appear at the source. This gate-source voltage will quickly enhance the MOSFET, minimizing the drain-source voltage drop. However, if the battery is installed backwards then the body diode will be reverse-biased and block the flow of current. This effectively prevents a reverse-voltage at the input from damaging the system [33].

3.4.2.2 Voltage Regulation

The voltage regulator portion of the power supply consists of four switching regulators, as shown previously in Figure 3.10. A PTR08060W switching regulator is used at the input to step the unregulated battery voltage down to 5.5V. This 5.5V power rail is then distributed to both the auxiliary board and three PTH04070W switching regulators on the main board. These secondary regulators are used to generate the 1.2V, 2.5V, and 3.3V digital supply voltages needed by the FPGA modules and other main board devices. The unregulated battery voltage and the 3.3 volt supply are also distributed to the auxiliary board.

The PTR08060W and PTH04070W are highly integrated, configurable switching regulator modules from Texas Instruments. Both devices allow the output voltage to be set using a single resistor. They require few external capacitors, which can simplify board layout and minimize the amount of area consumed. The PTR08060W can provide up to 6A of output current, while the PTH04070W can supply up to 3A. Both regulators feature undervoltage lockout, output short-circuit protection with automatic reset, and achieve typical efficiencies of 80 to 90 percent.

The PTR08060W and PTH04070W devices, like all switching regulators, exhibit two forms of noise at their inputs and outputs: ripple voltage and switching spikes. Ripple voltage occurs at the fundamental switching frequency of the regulator and can generally be attenuated by increasing the input/output capacitance or decreasing capacitor ESR. The switching spikes

are much higher frequency in nature (over 100 MHz) and cannot be reduced as easily due to the parasitic inductance (ESL) of the input/output capacitors [34]. This high frequency noise is problematic since it contributes to the EMI generated by the system and can interfere with the sensitive analog circuitry on the auxiliary board. In order to decrease this noise, "PI Filters" were added to the input and output of the voltage regulation stage. These filters are composed of a series ferrite bead surrounded by two ceramic bypass capacitors. Because ferrite beads exhibit increasing impedance at higher frequencies, while presenting minimal resistance at DC, this configuration has significantly improved spike attenuation compared to a simple capacitor-only filter [34, 35]. The PTR08060W regulator and associated PI filters are shown in Figure 3.12. Similar filters were used at the outputs of the PTH04070W regulators.

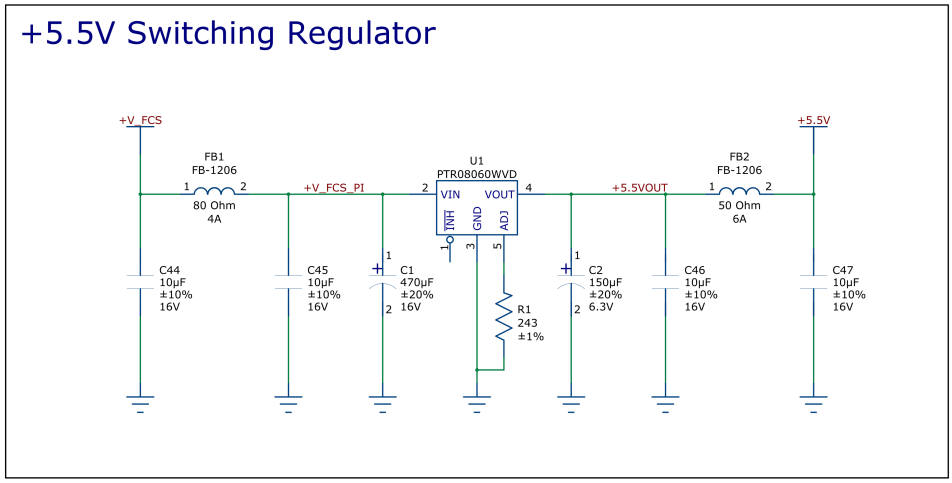

Figure 3.12: 5.5V Switching Regulator and PI Filters

The measured noise present at each regulator output, along with the noise at the system input, is presented in Table 3.2 below. Detailed voltage waveforms are provided in Figure 3.13 and Figure 3.14. A Tektronix MSO 4034 mixed signal oscilloscope was used to record all measurements. The PTH04070W regulators show slightly less noise than the PTR08060W, with only 4-7mV ripple and 16-23mV spikes. Although there is a relatively large ripple voltage at the

system input (12V Supply), the spike voltage is limited to only 10mV. Radiated EMI from the system power leads should therefore be minimal.

Table 3.2: Power Supply Noise

	DC Voltage	Ripple Voltage	Spike Voltage
1.2V Regulator	1.211V	4mVpp	23mVpp
2.5V Regulator	2.517V	4mVpp	16mVpp
3.3V Regulator	3.317V	7mVpp	17mVpp
5.5V Regulator	5.455V	13mVpp	26mVpp
12V Supply	11.90V	84mVpp	10mVpp

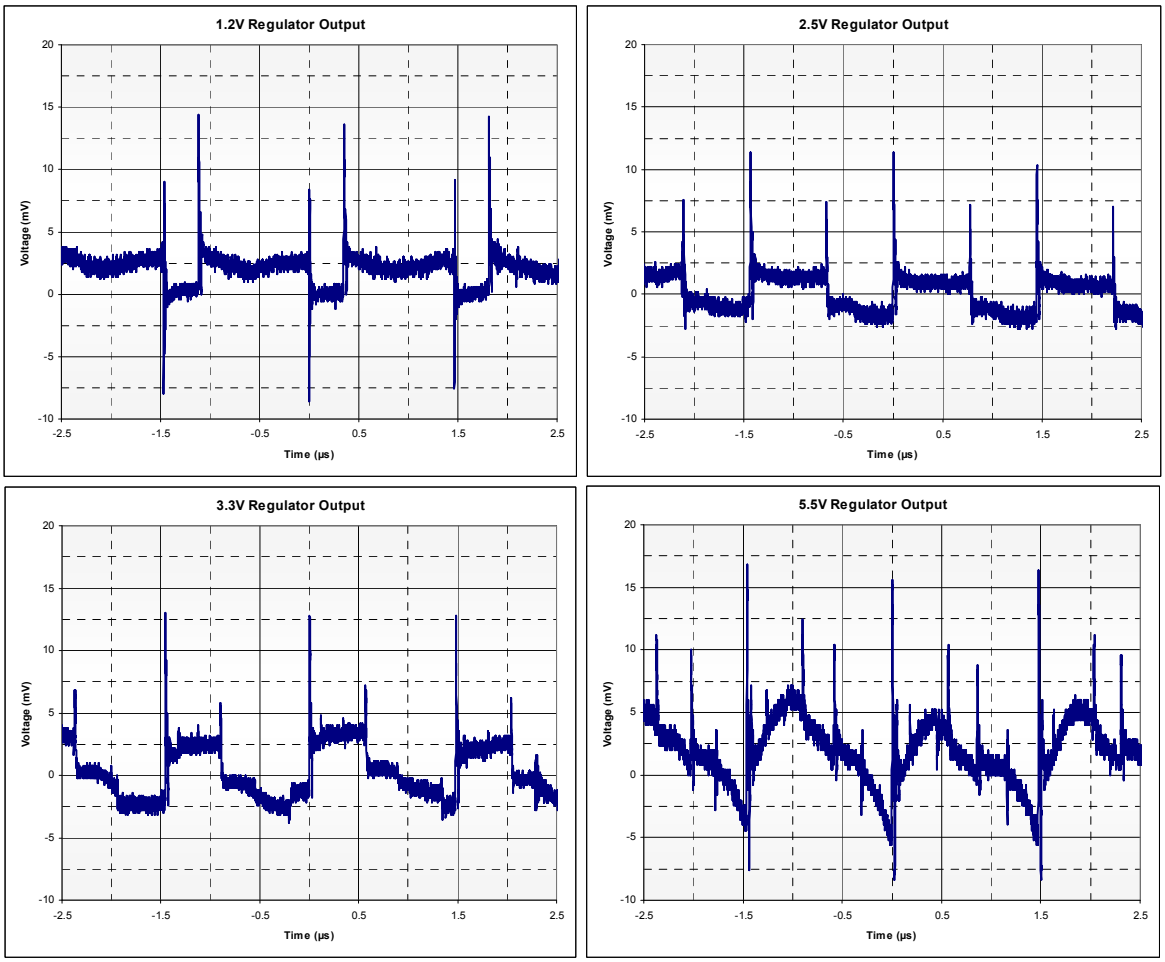

Figure 3.13: Switching Regulator Noise (1X Probe, AC Coupled)

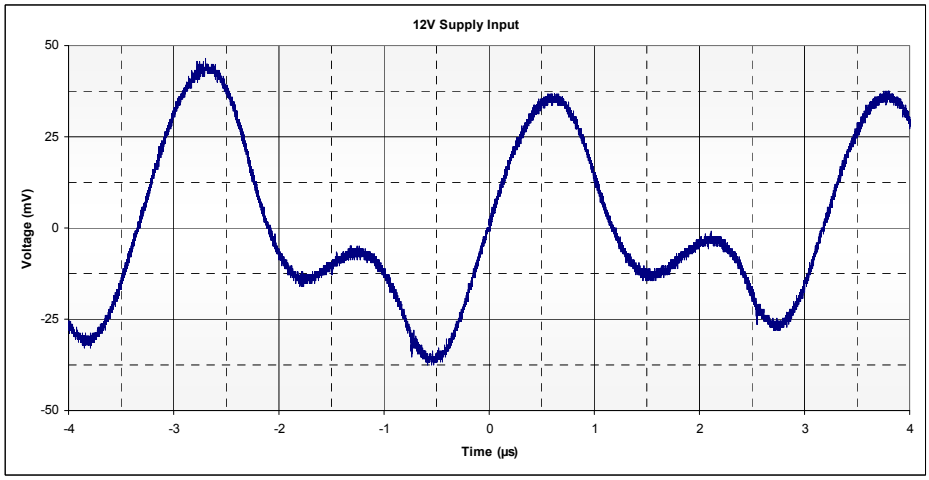

Figure 3.14: DC Supply Noise (1X Probe, AC Coupled)

3.4.2.3 Voltage Monitoring

The voltage monitoring stage provides undervoltage (brownout) detection and power-on-reset (POR) functionality for the two FPGA modules. This is accomplished using the ISL88021IU8FCZ Triple Voltage Monitor from Intersil. This IC monitors the three secondary power rails (1.2V, 2.5V & 3.3V) and triggers a reset condition whenever any of the three power rails drops below the limits specified in Table 3.3. Reset is only deasserted after all power rails remain above their respective thresholds for at least 200 ms.

Table 3.3: Undervoltage Limits

Power Rail	Trip Voltage
1.2V	1.10V
2.5V	2.32V
3.3V	3.09V

This functionality ensures that both FPGAs will always be correctly initialized at startup and will be reprogrammed in the event of a momentary power failure. Without this capability, an undervoltage condition could corrupt an FPGA's RAM contents and result in a lockup or other undesired behavior. This is especially important since the ICM FPGA controls the safety switch's mode select signal during normal operation. During reset the safety switch will also be

held in the manual control state, ensuring that recovery of the aircraft is always possible. More details will be provided in section 3.4.3 below.

3.4.3 Safety Switch

The main board features an integrated safety switch that allows for changing between manual and autonomous control of the UAV during flight. This is a critical safety feature that enables the human safety pilot to take direct control of the aircraft at any time, overriding the autopilot software. This is necessary in the event of an autopilot malfunction or failure, as well as during manual takeoff and landing.

The main board safety switch provides nine pulse-width modulation (PWM) input channels and eight PWM output channels. Eight of the input channels are used for servo control signals, while the ninth channel is used by the safety pilot to indicate manual or autonomous operation. When in manual mode, the safety switch will buffer the input signals from the RC receiver and pass them directly to the servo outputs. In autonomous mode, the autopilot control signals will be buffered and passed to the servo outputs. In both cases, all nine input channels will be buffered and connected to the ICM FPGA. All safety switch connector leads are soldered directly to the main board, eliminating one potential source of failure during flight.

A logic diagram of the safety switch is provided below, in Figure 3.15. Red signals and devices are powered from the RC flight battery, while blue signals and devices are powered by the FCS power supply. As shown, the output multiplexer is powered off of the RC flight battery. This is done so that the safety switch can remain functional even if the autopilot hardware loses power. It also has the effect of increasing the output signal amplitude to that of the RC battery, which has a nominal voltage of 4.8V. This increases the noise margin of the control signals compared to the typical output voltage of most RC receivers. It should be noted that the output

multiplexer can remain functional at battery voltages as low as 3.3V, which is significantly less than the servos and RC receiver are capable of functioning at.

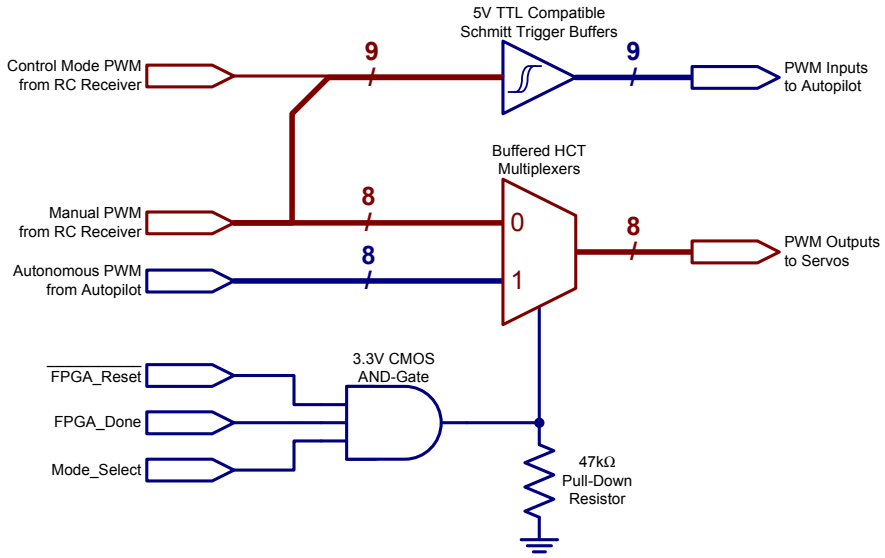

Figure 3.15: Safety Switch Logic Diagram

During normal operation, custom control logic in the ICM FPGA is used to monitor the PWM input signals and control the multiplexer at the output of the safety switch. If the ICM fails or loses power, the safety switch will be forced into manual mode. This is accomplished using the three-input AND gate and pull-down resistor shown in Figure 3.15. The AND gate takes in the FPGA Reset, FPGA Done, and Mode Select signals. The power supply voltage monitor will ensure that the Reset line is held low at startup and during undervoltage conditions. After reset, the FPGA Done signal will not go high until the FPGA is fully programmed and operational. Only when both signals are high will the ICM be able to put the safety switch in autonomous mode using the Mode Select signal. If the FCS completely loses power, then the pull-down resistor will ensure that the multiplexer stays in manual mode.

3.4.4 I/O Interfaces

In addition to the PWM signals used by the safety switch, the main board features several other digital interfaces that allow for communication with external hardware. A total of six

dedicated RS-232 serial ports are available. Two of these are connected to the FCM and four are connected to the ICM. These ports are typically used to interface with external IMU/INS devices, such as the Microbotics MIDG II or the Microstrain 3DM-GX3. A total of 24 general purpose I/O (GPIO) pins are also provided. Sixteen of these pins are directly connected to the FCM FPGA, while the remaining eight are connected to the ICM FPGA. These GPIO pins can be controlled through software or via dedicated logic in the FPGAs. This allows them to be customized to handle nearly any interface function compatible with 3.3V TTL signal levels.

Several GPIO and serial interface lines also exist between the main board and auxiliary board. These signal lines allow the ICM and FCM FPGAs to control and communicate with the application specific devices on the auxiliary board.

3.4.5 Non-Volatile Storage

Along with the I/O capabilities mentioned above, the main board also has an onboard microSD slot. This storage interface is connected to the FCM and substantially increases the amount of non-volatile memory available to the autopilot software. Standard microSD cards provide up to 2GB of memory, while high-capacity variants can accommodate up to 64GB.

The primary benefit of this increased storage is the ability to store longer and more detailed flight logs than was previously possible. This increased data capacity can facilitate a much more thorough analysis of the aircraft's flight performance. Also, since the storage is non-volatile, it can allow the mission data to be recovered in the event of a power-loss or crash. Another potential benefit is increased flexibility during development and testing. Multiple versions of the autopilot software and support utilities could be stored on the microSD card and be selected from at startup. This would allow for the testing of new FCS software, while still providing easily accessible backups of "known good" versions.

3.5 Auxiliary Board

The current version of the auxiliary board contains the analog and RF devices required by the autopilot system. The analog front end includes anti-aliasing low-pass filters, low-noise linear voltage regulators, and a high-precision analog to digital converter. Barometric pressure transducers for measuring altitude and airspeed information are integrated, while connectors are provided for interfacing with additional external analog sensors. Supported RF devices include 900MHz radio modem modules from Digi-MaxStream and GPS receiver modules from uBlox. These devices can be mounted directly on the auxiliary board, saving space.

3.5.1 PCB Design

The auxiliary board PCB is a four layer design measuring 3.8 inches by 2.9 inches. Signal traces and components are located on the top and bottom layers, while the inner layers are used for power and ground. Any unused areas of the outer layers are filled with copper and connected to ground using numerous stitching vias. The RF and analog components are located in physically separate areas of the board. The ground and power planes are also split, preventing the possibility of conducted EMI and noise coupling between the two sections. Like the main board, all bypass capacitors are placed close to the corresponding IC power pins and via-in-pad techniques are used when possible. The actual layer stack-up is shown in Table 3.4 and the board layout is shown in Figure 3.16.

Table 3.4: Auxiliary Board Layer Stack-Up

#	Layer Type	Primary Functions
1	Top Layer	Signal Traces, Components & Connectors
2	Inner Layer	Split Ground Plane
3	Inner Layer	Split Power Plane
4	Bottom Layer	Signal Traces, Components & Connectors

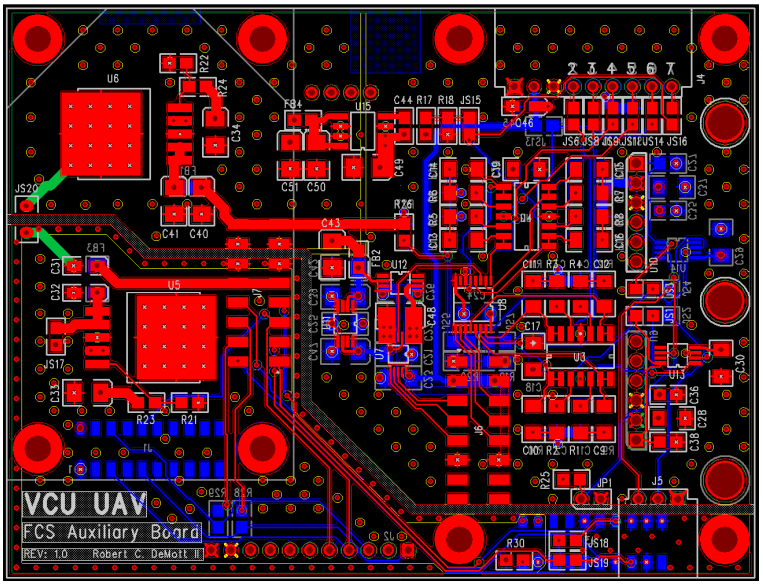

Figure 3.16: Auxiliary Board PCB Layout

Most ceramic capacitors are subject to piezoelectric effects, which means that mechanical stress and vibration can translate into electrical noise [36]. To prevent this from distorting the sensitive analog sensor readings, all ceramic capacitors on the auxiliary board are of the C0G/NP0 type. These capacitors utilize EIA Class 1 dielectric materials which have essentially zero piezoelectric effects. They also exhibit negligible temperature dependence compared to other ceramic capacitors [37]. However, their maximum capacitance rating is limited to approximately 0.1uF due to physical size constraints. In cases where higher capacitance is required, such as voltage regulator bypassing, tantalum capacitors are used.

3.5.2 Power Supply

Analog and RF circuits are highly sensitive to electrical noise and require clean, stable power for optimal performance. A switching supply can noticeably raise the noise floor, reducing the accuracy of analog sensor readings and decreasing the maximum range of radio communications [38]. Because the switching regulators on the main board are too noisy to be

used directly, low-noise linear regulators are used to power these sensitive electronics. A diagram of the auxiliary board power supplies is provided in Figure 3.17.

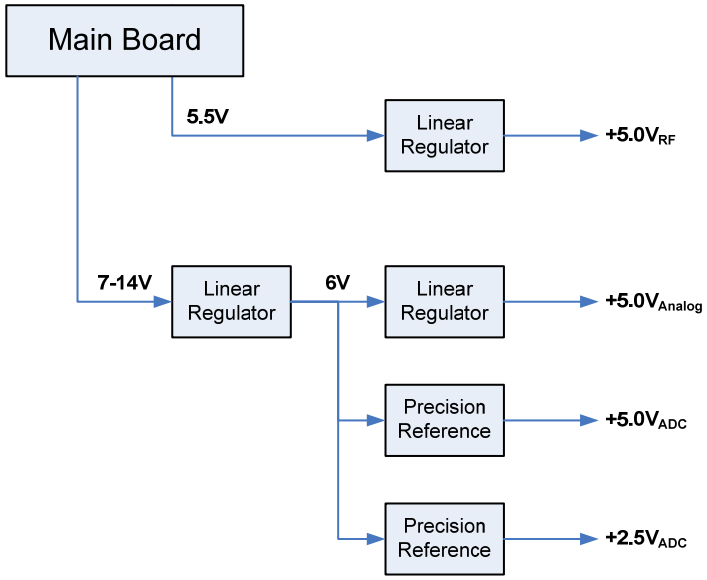

Figure 3.17: Auxiliary Board Power Supply

The onboard analog sensors and signal conditioning hardware require a total supply current of only 40 to 50 milliamps. When external analog sensors are factored in, the worst case power draw is approximately 150mA. These low power requirements allowed the analog supply to run directly off of the FCS battery without violating linear regulator thermal constraints or wasting too much power. This was beneficial due to the lower levels of high frequency noise present at the system input, as noted in section 3.4.2.2. As shown in Figure 3.17, the analog supply uses a two stage design. A primary regulator steps the battery voltage down to 6V, while secondary regulators power the ADC and analog electronics. This results in improved noise rejection and eases the thermal load placed on the secondary regulators.

The RF supply could not be powered directly from the battery due to the significantly higher power requirements. The radio modem requires approximately 730mA when using the highest power transmit setting, and the GPS unit uses 70mA in the worst case. This would place an excessive thermal load on the linear regulator and waste over 5 watts of power. Instead, a

linear regulator is used to post-regulate the 5.5V supply from the main board down to 5V. This achieves an efficiency of ~90%, dissipating less than 0.5 watts of waste heat in the worst case.

Most linear regulators generate little noise voltage at the output and typically have good input ripple rejection at low frequencies. However, their power supply ripple rejection (PSRR) tends to degrade significantly at higher frequencies. This means that high frequency spikes from switching regulators will pass directly through a linear regulator with negligible attenuation. This is important to note when post regulating a switching supply, or when a linear regulator is powered from the same source as a switching supply. In both cases, ferrite bead PI filters can be used augment the linear regulator, substantially improving its high frequency noise rejection [39]. This method was used successfully with all of the linear regulators and precision voltage references used on the auxiliary board. There were no discernable spikes at any of the analog supply outputs and any noise present was below the 1mV noise floor of the Tektronix oscilloscope. Noise at the RF supply output was limited to approximately 5mV.

3.5.3 Analog Front End

The auxiliary board contains hardware for facilitating the measurement of analog sensor signals. A top level diagram of the analog system is shown below in Figure 3.18. This system is composed of two main sections: analog signal conditioning and analog data capture. The signal conditioning stage contains input buffers and active low pass filters for each analog input channel. It provides anti-aliasing and noise rejection prior to the data capture stage. The data capture stage consists of an 8-channel analog multiplexer and a 16-bit analog-to-digital converter. Multiplexer channel selection and sample conversion are controlled using custom VHDL logic in the ICM FPGA. See section 4.3 on page 70 for more details on the FPGA hardware.

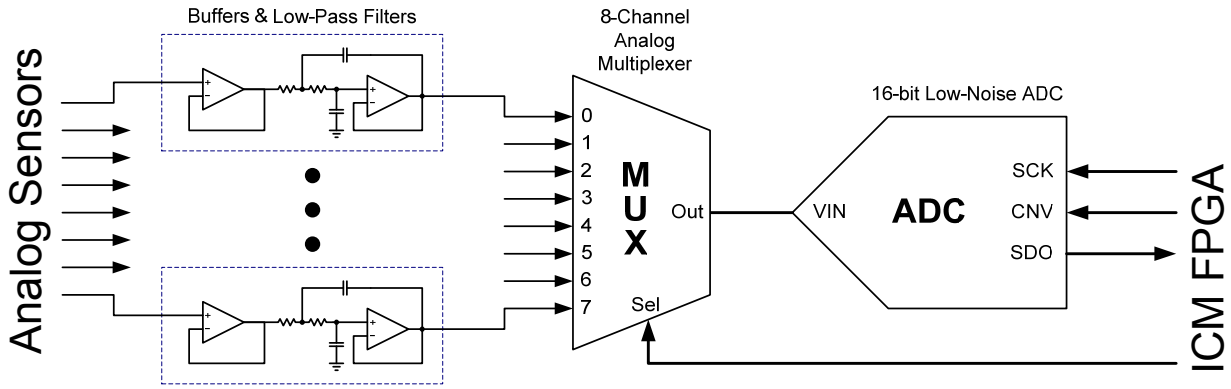

Figure 3.18: Analog Front End Hardware

The system supports capturing data from up to 8 analog sources and accepts signals in the 0 to 5V range. Each input channel can be connected to either a predetermined onboard device or an external sensor. Optional voltage dividers are provided on most inputs to allow measuring of sources greater than 5V, such as the FCS and RC batteries. The default input configuration is shown in Table 3.5.

Table 3.5: Analog Input Configuration

Channel	Default Connection
0	Absolute Pressure Sensor
1	Differential Pressure Sensor
2	FCS Battery Voltage
3	RC Battery Voltage
4	Open/External
5	Open/External
6	Open/External
7	5.0V Analog Regulator

3.5.3.1 Analog Data Capture

An 8-channel, high-speed analog multiplexer is used to select between the filtered analog input signals. The device used for this purpose is the MAX4617 low voltage CMOS analog multiplexer from Maxim-IC. This multiplexer features low crosstalk (-96dB), high off-isolation (-93dB), and low distortion (0.017%). It has a maximum on-resistance of 10Ω with 1Ω matching between channels and a fast switching time of 15ns [40]. This combination of fast switching time and low on-resistance ensures that the filter op-amps can quickly charge the

ADC's input capacitor after a channel transition occurs. After multiplexer channel switching, the filter outputs typically settle to within one 16-bit code in less than 0.5 μs. The worst case settling time observed was 0.8 μs, which allows for channel transition rates of up to 1.25 MHz.

The output of the multiplexer is connected to an Analog Devices AD7980. The AD7980 is a 16-bit successive approximation ADC capable of sampling at rates up to 1 MHz. It communicates with the host via a high-speed SPI bus and allows for daisy-chaining of multiple ADCs. The AD7980 features a worst-case integral non-linearity (INL) of ±1.25 LSB, a signal to noise and distortion (SINAD) ratio of 91.25dB, and a total harmonic distortion (THD) of -110dB. Maximum power consumption is 7.0mW at a sample rate of 1 MHz [41]. The AD7980's high SINAD and low THD ratings ensure a low noise contribution, while the low INL provides a high degree of accuracy. Its low power consumption allows it to operate from low-current, low-noise voltage regulators. In this case, the AD7980 actually operates from two high-precision voltage references which provide the ADC with superior PSRR. A Texas Instruments REF5025 is used for the ADC's core supply, while a REF5050 is used as the ADC's precision 5V reference.

When the ADC is operated at its maximum rate of 1 MHz, the effective per-channel sample rate is 125 kHz due to the 8-channel multiplexer. Although this may seem excessive compared to the FCS update rate of 200 Hz, it offers two important benefits. The primary benefit is that it relaxes the analog filter requirements. In order to avoid signal distortion due to aliasing effects, any frequency content higher than the Nyquist frequency must be attenuated prior to sampling. This is accomplished by using analog low pass filters at the input of each channel to band limit the signals. With a sampling rate of 125 kHz, the Nyquist frequency is equal to 62.5 kHz. Noise above this frequency can easily be attenuated with a simple two-pole active filter, while still preserving information within the FCS bandwidth of 100 Hz.

Significantly lower sampling rates would either sacrifice system bandwidth or require higher order analog filters to prevent aliasing. Sampling faster therefore allows analog filter complexity to be reduced, which in turn saves cost and board area. The sampled data can then be digitally filtered and down-sampled to the required FCS rate using logic in the ICM FPGA.

Not only does oversampling relax the anti-aliasing filter requirements, it can also be used to enhance the resolution of the sampled data by reducing quantization noise. By combining oversampling with averaging and decimation, the effective number of bits (ENOB) and signal-to-noise ratio (SNR) of the sampled data can be increased. The maximum increase in ENOB is given by $N^{+} = \log_4(F_{os}/F_s)$, where F_{os} is the oversampling frequency and F_s is the minimum sampling rate necessary to satisfy the Nyquist sampling criterion [42]. In this case, F_{os} = 125 kHz, and F_s = 200 Hz. Therefore, this method could theoretically increase the sample resolution by approximately 4.64 bits. In order for this method to work, there must be some noise at the input with sufficient amplitude to toggle at least one LSB. This noise must also be approximately white and uncorrelated with the input signal. If these conditions are not satisfied, this method will be less effective. Nevertheless, it can be a very useful technique and should be investigated in cases where higher resolution would prove beneficial.

3.5.3.2 Analog Signal Conditioning

Each analog channel contains a low pass filter preceded by an input buffer stage. The buffer is simply an op-amp configured as a unity gain voltage follower. This input buffering stage is used to prevent the filter stage from loading the source device or interfering with its operation. Without the buffer stage, a sensor's source impedance would combine with the low pass filter and distort the frequency response. The AD8608 op-amp from Analog Devices was chosen for the input buffer. The AD8608 is a unity gain stable, low noise, quad op-amp device

featuring rail-to-rail input and output (RRIO). It has a maximum offset voltage of 65μV, a maximum input bias current of 1pA, and a gain-bandwidth product (GBP) of 10 MHz. This combination of attributes makes it well suited for sensor buffering applications.

The op-amp chosen for the low-pass filter was the AD8618 from Analog Devices. This device is a precision quad op-amp with unity gain stability and RRIO. It has a maximum offset voltage of 65μV, a maximum input bias current of 1pA, and a GBP of 20 MHz. It also has an output drive strength of 150mA and a slew rate of 12V/μs. These features are important since they allow the filter output to stabilize quickly when its multiplexer channel is selected.

The 2-pole Sallen-Key active filter topology was chosen due to its relative simplicity and non-inverting, unity-gain characteristics. The Texas Instruments FilterPro v2.0 software was used to calculate the necessary component values. A Bessel filter response was chosen due to its maximally flat group delay and linear phase response. This is important to minimize distortion in the time-domain due to overshoot and ringing [43]. The low pass cutoff frequency was set to 250 Hz. This should theoretically yield a filter with an attenuation of -91.5 dB at the Nyquist frequency of 62.5 kHz, which is below the SINAD of the ADC. This means that aliasing should be completely prevented, even in the presence of nearly full-scale noise.

Ideally, the resistor values should be limited to approximately 1kΩ to minimize noise. This is because Johnson-Nyquist (thermal) noise increases with increasing resistance. However, the use of C0G capacitors limits the maximum capacitance value to 0.1μF. This in turn necessitates larger resistance values to achieve the target filter response. Since vibration is unavoidable in this application, it was decided that larger resistance values could be tolerated in exchange for preventing piezoelectric effects. The calculated component values and filter response from FilterPro are shown below in Figure 3.19.

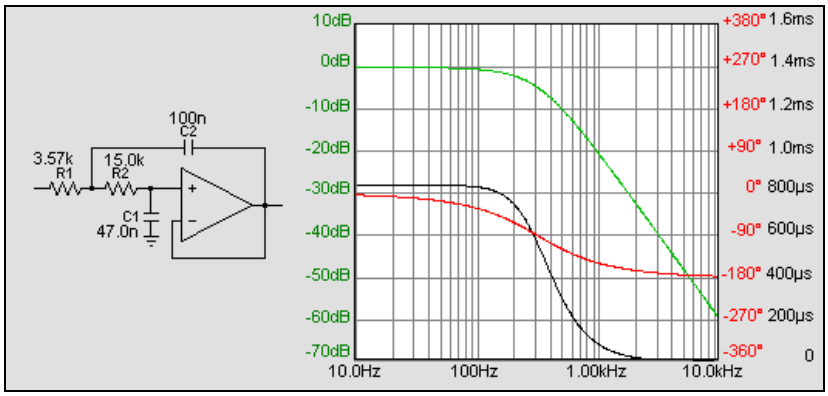

Figure 3.19: Design of Anti-Aliasing Low-Pass Filters: Component Values (left), Amplitude Response (right, green), Phase Response (right, red), Group Delay (right, black).

Unfortunately, the frequency response of the actual filter doesn't completely match the ideal response calculated by the software. This is due to non-ideal characteristics inherent to all real world op-amps. After the -3dB point, the frequency response of the filter should continue to roll off at -40 dB per decade indefinitely. In practice, the frequency response actually stops dropping and begins to rise at 20 dB per decade past ~50 kHz. It then peaks at around 15 MHz before dropping again at -20 dB per decade. This effect was confirmed by both in lab testing and simulation. The frequency response of the simulated filter is shown below in Figure 3.20.

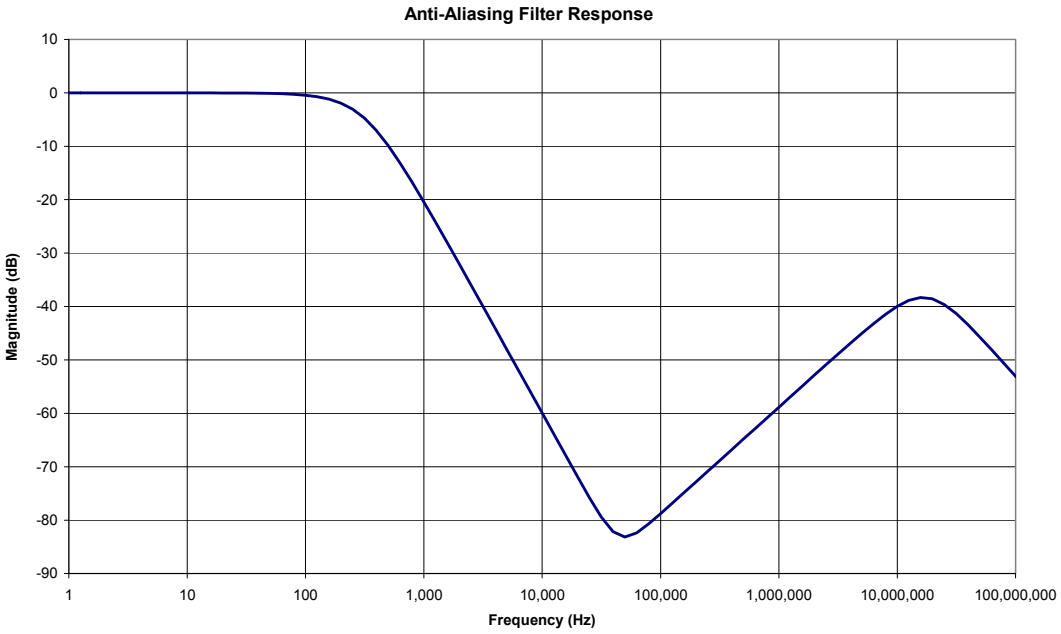

Figure 3.20: Simulated Frequency Response of Anti-Aliasing Filters

This effect is known as stop-band leakage and occurs due to the amplifier's output impedance. As frequency increases, the open-loop gain of the amplifier decreases resulting in increased output impedance. This combines with the feedback capacitor and essentially forms a high-pass filter. The frequency response begins to rise when the output impedance becomes larger than resistor R1. This effect stops once the open-loop gain of the op-amp becomes zero [44, 45]. The limited bandwidth of the input buffer then causes the frequency response to begin decreasing again.

Choosing an op-amp with a higher gain-bandwidth product (GBP) could potentially reduce this effect. Increasing the resistor values and decreasing the capacitor values would also decrease stop-band leakage by increasing the cutoff frequency of the parasitic high-pass filter. This would increase offset voltage and contribute additional thermal noise from the resistors, however. Another possible way to mitigate this effect would be the addition of a simple first order RC filter at the input. A single pole 25 to 50 kHz low pass filter would cancel out the stop band leakage, causing the frequency response to temporarily level out around -90db instead of rising. This effect has not been an issue in practice, however, due to the relatively low noise amplitude observed in the affected frequency range.

3.5.4 Barometric Pressure Sensors

The auxiliary board contains two barometric pressure sensors for measuring altitude and airspeed information. The particular devices used are the MPX5010DP and MPX5100AP from Freescale Semiconductor. Both sensors operate from the 5V analog supply and translate pressure to voltage using piezo-resistive transducers. They are temperature compensated and operate over the range of -40° to +125° C. These sensors have a useful mechanical response up to approximately 500 Hz. However, their noise output can extend up to 1 MHz with amplitudes of

up to ±10mV [46]. The analog signal conditioning on the auxiliary board successfully filters out the majority of this noise. Residual noise is further attenuated using digital filtering in the ICM FPGA. The combined analog and digital filtering can increase the accuracy of the measurements by approximately two orders of magnitude versus the unfiltered measurements.

The MPX5010DP is a differential pressure sensor capable of detecting pressure differences of up to 10 kPa. Pressure is converted to voltage with a linear relationship of the form: $V_{out} = V_S \cdot (0.09 \cdot P + 0.04)$, where P is the pressure differential in kilopascals and V_S is the source voltage (5V). This sensor is connected to the aircraft's Pitot tube, allowing measurement of airspeeds up to 243 knots. The details of the impact pressure to airspeed conversion will be covered in Chapter 5.

The MPX5100AP is an absolute pressure sensor capable of measuring pressures between 15 and 115 kPa. It has a similar linear transfer function, of the form: $V_{out} = V_S \cdot (0.009 \cdot P + 0.095)$, where P is the static pressure in kilopascals and V_S is the source voltage (5V). Altitudes between -3500 and +44000 feet AMSL can be calculated using the static pressure information. The previous autopilot system utilized additional offset and gain circuitry to increase the resolution of these static pressure measurements. Unfortunately, this limited the maximum altitude to approximately 2500 feet. This gain stage is no longer necessary due to the improved signal conditioning and digital filtering of the new autopilot platform. The full altitude range of the sensor is now usable with substantially higher resolution. Accuracy is also improved, typically within ±1 ft of the true barometric altitude. Additional details will be provided in Chapter 5.

3.5.5 GPS Receiver

The onboard GPS module is an RCB-4H Antaris-4 programmable GPS receiver module from uBlox. This model is a 4Hz, 16-channel receiver with a 2-D accuracy of 2.5m and 3-D

accuracy of 5m. It can operate from a low-noise supply between 3.15V and 5.25V. Typical current draw is 39mA and maximum current draw is 70mA. It has dual TTL serial interfaces, each capable of operating at speeds between 9600 baud and 115200 baud. Data can be output using standard NMEA strings or UBX binary packets. It is compatible with both active and passive external antennas and has a tracking sensitivity of -158 dBm. This GPS receiver is optional if an external INS/GPS device such as the Microbotics MIDG II is used.

3.5.6 Wireless Data Link

A 9XTend radio modem from Digi-MaxStream can be directly mounted on the auxiliary board. This modem operates in the 900 MHz band and supports continuous transfer rates up to 115200 bps. It has a line-of-sight range of 7 miles when using a dipole antenna, and up to 20 miles when high-gain directional antennas are used. Transmit power is variable between 1mW and 1000mW. It can be powered from supplies between 2.8V and 5.5V; although the highest transmit setting requires at least 4.75V. Current consumption is dependent on transmit power, supply voltage, and bandwidth utilization. The modem draws 270mA @ 5V when transmitting at 100mW, while transmitting at 1000mW requires 730mA @ 5V.

It communicates with the autopilot using a TTL serial interface, and supports several operating modes. In transparent mode it can act as a simple serial link replacement between two nodes. In API mode it supports direct addressing and frame-based communication between multiple nodes. When DigiMesh mode is enabled, several nodes can form a self-healing mesh network with automatic route discovery and recovery. Currently, the transparent mode of operation is used in order to maintain compatibility with the existing ground station software. Telemetry update rates of over 30 Hz are achievable when using this radio link.

Chapter 4: ICM FPGA Logic

This chapter will cover the specific FPGA logic utilized in the Instrumentation Control Module. An overview of the system architecture will be provided in Section 4.1. Section 4.2 will discuss the Xilinx IP used in this design. The remaining sections will describe the custom FPGA IP cores created by the author for use in the ICM. A discussion of each core's capabilities and logic implementation details will be provided. Any Spartan-3 features utilized will be mentioned, as well as the FPGA resources required for each core.

4.1 FPGA Logic Overview

The FPGA used in the ICM is a Xilinx Spartan-3 XC3S400. This FPGA provides the equivalent of 400,000 system gates worth of programmable logic. It contains 896 configurable logic blocks (CLBs), with each CLB being split into 4 "slices". Each CLB slice contains two flip-flops, two 4-input lookup tables (LUTs), two wide-function multiplexers, and a dedicated carry-chain. This FPGA also contains 16 dedicated 18x18 multipliers, 56kb of distributed RAM, and 288kb of block RAM [47]. These programmable logic resources provide a high degree of flexibility.

In the ICM, these reconfigurable elements were used to realize various interfaces for connecting to sensors, actuators, and communications hardware. FPGA logic was also used to implement a soft-core processor and provide hardware acceleration capabilities. A combination of Xilinx provided IP and custom user logic was employed in this design. A top-level diagram of

the FPGA logic and the connections to external hardware is shown in Figure 4.1. In this figure, Xilinx cores are shown in white and custom cores are shown in a darker shade of blue.

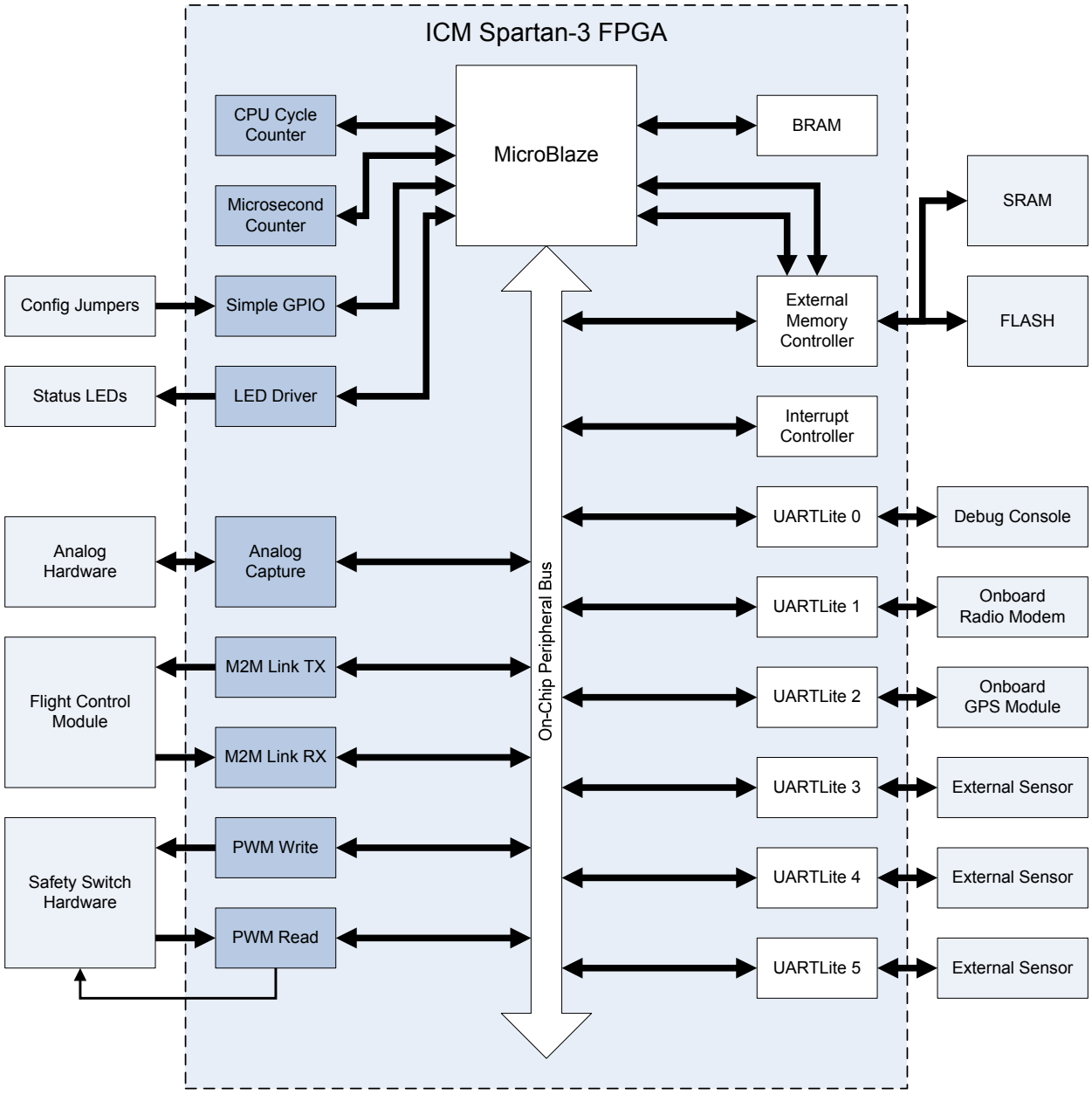

Figure 4.1: Top Level Diagram of ICM Spartan-3 FPGA Logic

As seen in the above diagram, this design is based around the Xilinx MicroBlaze soft-core processor. Although this is the same CPU architecture used in the previous generation of autopilot hardware, this system achieves higher computational performance. Specifically, the MicroBlaze in this system achieves a performance of 37.5 DMIPS. This is approximately 80%

of its theoretical performance at 50 MHz, and an improvement of 4.4 times compared to the previous generation hardware. This is mainly due to the lower latency memory controller interface and CPU cache settings chosen. More details will be provided in the following section.

The main custom cores in this design are the Analog Capture Core, the Pulse Width Modulation (PWM) Cores, and the Module-to-Module (M2M) Link Cores. The Analog Capture Core is responsible for interfacing with analog sensor devices and digitally filtering the captured data. The PWM cores are used to read the manual control signals from the RC receiver and generate autonomous control signals to drive the aircraft servos. The M2M Link cores provide an interface for exchanging data between the ICM and FCM. Other cores were also created to handle simpler tasks, such as general purpose I/O and system timers.

These custom cores were developed using the Xilinx Integrated Synthesis Environment (ISE) 10.1 and Xilinx Embedded Development Kit (EDK) 10.1 software [48, 49]. Xilinx documentation, such as the "XST User Guide", was consulted to ensure effective VHDL coding practices were followed [50, 51, 52]. All custom cores were simulated using the Mentor Graphics ModelSim SE 6.5a software package to verify correct operation [53].

4.2 Xilinx IP Cores

The Xilinx IP cores utilized in the ICM design will be briefly described in the following subsections. These cores were provided with the Xilinx Embedded Development Kit (EDK) 10.1 software. Specific configuration parameters and design choices that affect system performance or FPGA logic utilization will be mentioned.

4.2.1 Xilinx MicroBlaze Core

The MicroBlaze is a 32-bit RISC soft-core processor optimized for use with Xilinx FPGA devices. It features a modified Harvard architecture with 32 general purpose registers and a

single-issue pipeline. The core itself is highly configurable, allowing for additional capabilities and improved performance at the cost of increased logic utilization in the FPGA. Optional instructions that can be enabled include: integer multiply, integer divide, barrel shifter, and pattern compare. An FPU can also be added to accelerate floating point math operations [54]. A block diagram of the MicroBlaze is shown below in Figure 4.2.

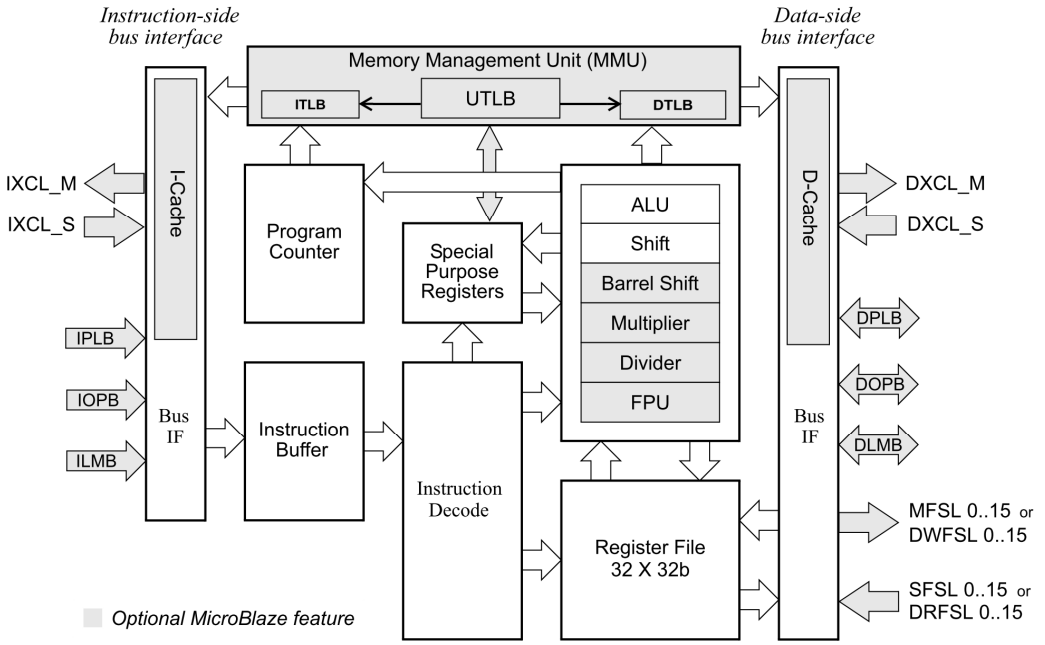

Figure 4.2: MicroBlaze Core Block Diagram [54]

The MicroBlaze configuration used in the ICM Spartan-3 is shown below in Table 4.1. Both instruction and data caches were enabled to improve performance. The Xilinx CacheLink (XCL) interfaces were also utilized to provide a direct link between the MicroBlaze and the memory controller. This lowers memory access latency and increases throughput. Although not strictly necessary, the floating point unit was also included. If additional FPGA resources are needed in the future, the FPU could be disabled without adversely affecting performance. This would free up an additional 15% of the total FPGA resources.

Table 4.1: ICM MicroBlaze Configuration

Processor Version	7.10.d
Operating Frequency	50 MHz
Implementation	Area-Optimized
Hardware Barrel Shifter	Enabled
Hardware Integer Multiply	Enabled, 32-bit
Hardware Integer Divide	Enabled
Hardware Pattern Comparator	Disabled
Floating Point Unit	Enabled, Basic
Memory Management Unit	Disabled
Instruction Cache	Enabled, 8kB
Instruction CacheLink Interface	Enabled
Data Cache	Enabled, 4kB
Data CacheLink Interface	Enabled

4.2.2 Xilinx MicroBlaze Bus Interfaces

The MicroBlaze supports several different bus interfaces for connecting to peripheral cores. These interfaces are the On-Chip Peripheral Bus (OPB), Processor Local Bus (PLB), and the Fast Simplex Link (FSL) [54, 55, 56, 57].

The OPB is a simple, fully synchronous bus supporting multiple masters and slaves. The MicroBlaze and peripheral cores communicate using 32-bit memory mapped I/O. The bus itself is essentially a distributed multiplexer and requires minimal logic resources in the FPGA. This bus is well suited to smaller designs and those running at frequencies below 100 MHz. Bus access latency is typically between 3 to 6 clock cycles depending on the configuration of the MicroBlaze and OPB arbiter.

The PLB is similar to the OPB, but utilizes a more complex set of communication signals. Like the OPB, it utilizes 32-bit memory mapped I/O and supports multiple masters and slaves. It can also operate at higher clock speeds and support larger designs than the OPB. This is accomplished by fully registering the data and address lines at every master and slave peripheral. As a consequence, this bus requires additional FPGA resources compared to the OPB and access latency is increased. Write operations typically require 6 to 10 clock cycles, while

read operations take 7 to 11 clock cycles. This added latency can be partially mitigated if burst transfers are used.

The FSL bus is a dedicated 32-bit point-to-point connection between two IP cores. The standard implementation features a built-in FIFO that allows for both synchronous and asynchronous communication. This interface supports high-throughput streaming I/O and is commonly used to accelerate CPU intensive algorithms.

A variant of the FSL bus, known as Direct FSL, omits the FIFO and directly connects the two IP cores. This is useful in cases where no additional buffering is needed since it requires extremely few FPGA resources. This configuration can achieve single clock cycle latency for both read and write operations.

In this design, custom cores that utilize memory mapped I/O were interfaced using the OPB. This was done to minimize logic usage and maximize throughput since the MicroBlaze does not support burst transfers. However, PLB versions of these cores were also created to allow for compatibility with the Virtex-4 PowerPC, which does not directly support the OPB. Simple cores that do not require memory mapped I/O were interfaced using the direct FSL bus. All bus interfaces operate at 50 MHz and are synchronous with the MicroBlaze CPU.

4.2.3 Xilinx External Memory Controller

The Multi-Channel OPB External Memory Controller (MCH OPB EMC) provides an interface between the OPB and up to four external memory devices. The core is fully parameterized and supports simultaneous control of multiple varying types of memory. Both synchronous and asynchronous memory devices with data widths of 8, 16, and 32-bits are supported. Data-width matching is supported and can be enabled on a per-device basis. The core also allows for a direct connection to the MicroBlaze caches via the XCL interface [58].

In this design, the MCH OPB EMC is used to connect to the SRAM and Flash memory on the ICM Spartan-3 Mini Module. The Flash memory is used to hold the ICM software image and the ICM config file. At boot-up, the ICM software is copied from Flash into SRAM and executed. The SRAM is then used for instruction and data storage during program execution. The XCL interfaces are used to provide low-latency burst access between the SRAM and the MicroBlaze.

4.2.4 Xilinx Interrupt Controller

The MicroBlaze only provides a single external interrupt request line. In order to support multiple interrupt sources, the OPB Interrupt Controller is used. This core is a simple programmable interrupt controller that supports up to 32 prioritized interrupt inputs. Several memory mapped registers are provided to enable and disable individual interrupts, detect pending interrupts, and acknowledge serviced interrupts [59].

The interrupt configuration used in this design is shown below in Table 4.2. Lower numbered interrupts have higher priority.

Table 4.2: ICM Interrupt Vector

IRQ	Peripheral Core
0	Analog Capture
1	M2M Receive
2	UARTLite 0
3	UARTLite 1
4	UARTLite 2
5	UARTLite 3
6	UARTLite 4
7	UARTLite 5

During testing, it was discovered that the OPB Interrupt Controller would occasionally generate spurious interrupts. Although this condition can usually be detected and handled in

software, it can nevertheless lead to degraded performance. This bug was corrected by modifying the VHDL source code provided by Xilinx.

4.2.5 Xilinx UARTLite

The OPB UARTLite is a simple UART core used to interface with off-chip asynchronous serial devices. This core is parameterized and allows for configurable baud rate, number of data bits, and parity modes. It supports full-duplex operation and includes 16-byte FIFOs for both receive and transmit channels [60].

In this design, several UARTLite cores are utilized to interface with various sensors and communications devices. The particular serial configuration used is shown below in Table 4.3. By default, all UARTs are configured for 115200 baud, 8 data bits, and no parity. These parameters may need to be adjusted depending on the specific requirements of the external sensors used.

Table 4.3: ICM UART Device Configuration

UART	Baud Rate	Serial Device	Logic Level
0	115200	User Debug Console	RS-232
1	115200	On-Board Radio Modem	3.3V TTL
2	115200	On-Board GPS Module	3.3V TTL
3	115200	External Sensor	RS-232
4	115200	External Sensor	RS-232
5	115200	External Sensor	RS-232

4.3 Analog Capture Core

A custom FPGA IP core was created to interface with the external analog to digital conversion circuitry and automate the process of capturing analog sensor readings. The core communicates directly with the ADC and controls the sampling process, allowing for continuous high-speed capture of analog data. It also features a built-in digital low-pass filter with decimation capability. This filter is used to reduce residual high frequency noise in the analog

data and then automatically down sample it to a rate usable by the FCS software. The filtered samples are stored in dedicated memory that can be accessed by the ICM software via the local peripheral bus (OPB or PLB). Because the sampling and filtering occur in hardware, a significant burden is removed from the ICM processor.

The core itself consists of several interconnected modules. These modules include the ADC Controller, CIC Decimator (low-pass filter), and Unsigned Divider (gain normalization) stages, as well as the sample memory and bus interface logic. A top level diagram showing the aforementioned modules is shown in Figure 4.3.

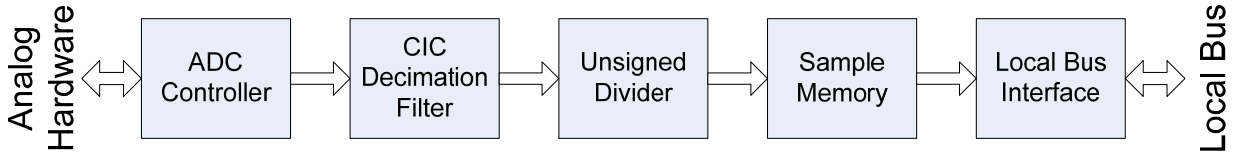

Figure 4.3: Analog Capture Core

The following sections will describe this IP core and its various subcomponents in detail. Information will also be provided on the ADC interface and the digital filtering techniques utilized.

4.3.1 ADC Interface

As stated in Chapter 3, the filtered analog sensor signals are fed into a MAX4617 8-to-1 analog multiplexer. The output of the multiplexer is then connected to the input of an AD7980, a 16-bit successive approximation ADC. The analog multiplexer channel is selected using a simple 3-bit binary signal that directly corresponds to the desired channel number. The ADC is controlled using a 3-wire serial interface that is compatible with the SPI protocol. The digital control and data signals of both the multiplexer and ADC are connected to the ICM FPGA, allowing the custom IP core to sample analog sensor data for all 8 input channels.

The ADC protocol utilizes three signals: CNV, SDO and SCK. CNV is the conversion start input signal, SDO is the serial data output signal, and SCK is the serial data clock. An ADC conversion sequence is initiated by the rising edge of the CNV signal. The CNV signal must then remain high for the duration of the conversion process (710 ns). After the conversion is complete, CNV can be brought low. This triggers the beginning of the acquisition phase. Once this happens, the most significant bit of the sample is output onto SDO. The remaining data bits are then clocked out by subsequent SCK falling edges. After the 16th SCK falling edge or when CNV goes high, whichever is earlier, SDO returns to high impedance and the acquisition phase is complete [41]. A timing diagram illustrating this process is shown below in Figure 4.4.

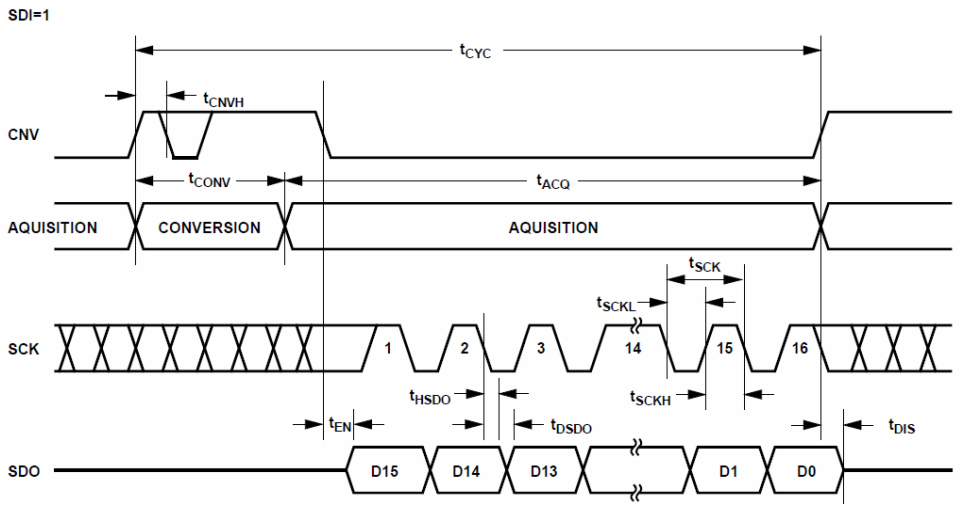

Figure 4.4: Timing diagram for the AD7980 3-wire CS mode without busy indicator [41]

In order to operate the ADC at its maximum sample rate of 1 MHz, the acquisition phase must complete within 290 ns. This requires a minimum SCK clock frequency of 55.17 MHz. A 60 MHz clock signal was generated by the ICM FPGA and used for this purpose. This clock signal was synthesized from the 50 MHz system clock using a Digital Clock Manager (DCM). This 60 MHz clock is also used for all logic in the analog capture core, with the exception of the bus interface logic which runs off of the 50 MHz system clock.

4.3.2 ADC Controller Logic

The ADC Controller is responsible for controlling both the ADC and multiplexer on the auxiliary board and transferring the captured samples into the digital low-pass filter. It primarily consists of a simple finite state machine (FSM), a binary up-counter and a shift register. Each state of the FSM lasts a fixed number of clock cycles and state transitions are triggered by the binary up-counter. The serial clock frequency, desired ADC sample rate and multiplexer width are specified at compile time using VHDL generics. These parameters are used to calculate the number of clock cycles for each state and to verify that ADC timing requirements are met.

The FSM itself is a three state Mealy machine with the following states: STARTUP, CONV_WAIT, and ACQ_SHIFT. A state diagram is shown below in Figure 4.5.

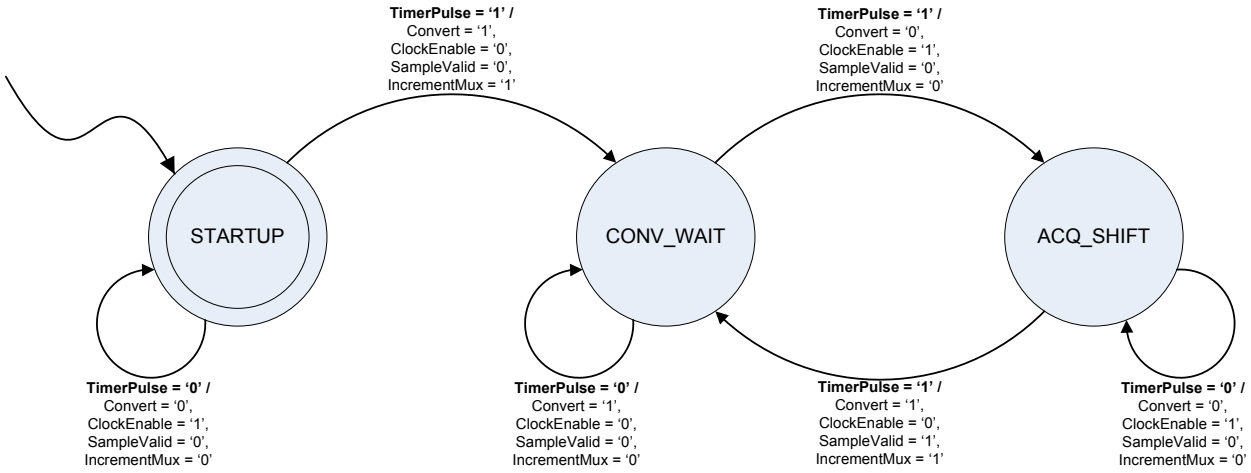

Figure 4.5: ADC Controller Finite State Machine

The STARTUP state is used immediately after reset to ensure that the ADC is initialized and in a known state. The FSM stays in this state for at least 1000 ns (the minimum sample period). It then transitions into the CONV_WAIT state.

The CONV_WAIT state is used to instruct the ADC to begin converting by raising the CNV signal. The FSM remains in this state for a minimum of 710 ns to ensure that the ADC's conversion stage has time to complete. The serial clock, SCK, is held low during this state to

minimize digital switching noise while the ADC is sampling. Also, the analog multiplexer channel is incremented one clock cycle after this state is entered. This exploits the ADC's built-in sample and hold circuitry, which makes it possible to change the ADC input after a conversion is started without affecting the result. By pipelining the process in this manner, the multiplexer output has additional time to settle before the next conversion phase begins.

The ACQ_SHIFT state is used to retrieve sample data from the ADC. During this state, the CNV signal is brought low and the serial clock is enabled. Sample data is shifted out of the ADC and shifted into an internal shift register. At the end of this state, a complete sample is stored and the FSM returns to the CONV_WAIT state.

As mentioned previously, an up-counter is used to control state transitions. The output of the counter is continuously compared with a pre-calculated value based on the current state of the FSM. When the counter reaches the compare value, a pulse is generated. This pulse automatically clears the counter and instructs the FSM to transition states. The state transition then causes a new compare value to be loaded. A detailed diagram of the ADC Controller logic is shown in Figure 4.6.

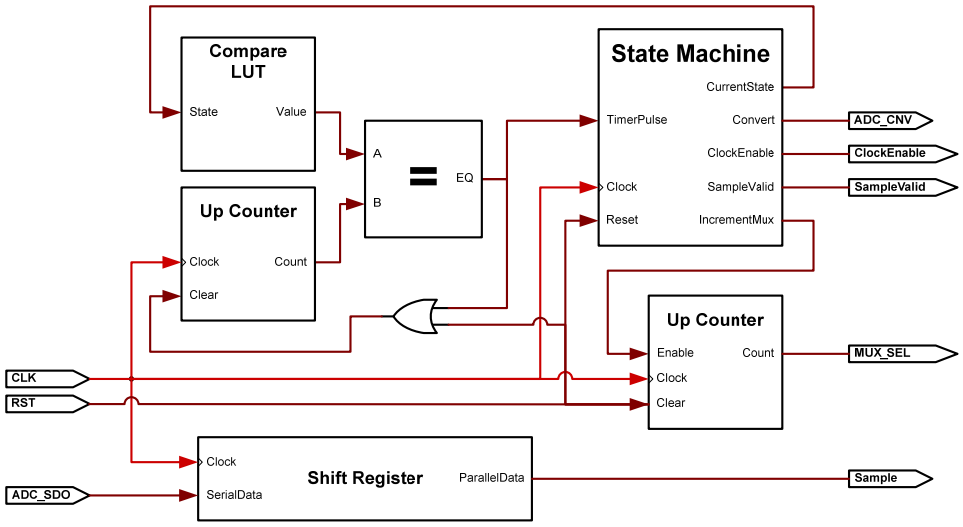

Figure 4.6: ADC Controller Logic Diagram

Lab testing showed that changing the analog multiplexer channel resulted in a worst case settling time of approximately 800 ns. Since this is less than the ADC's minimum sampling period, the ADC Controller was configured to run at the ADC's maximum sample rate of 1 MHz. This results in a per-channel sample rate of 125 kHz due to the 8 channel multiplexer.

Because of its relatively simple design, this portion of the Analog Capture Core requires very few FPGA resources. The resource utilization for the chosen configuration is shown below in Table 4.4. It uses only 21 slices, approximately 0.5% of the total resources available in the ICM's Spartan-3 FPGA.

Table 4.4: ADC Controller FPGA Resources

ADC Controller Parameters			Spartan-3 Resources			
Clock Frequency	Sample Rate	Mux Bits	Slices	Flip-Flops	LUTs	BRAM
60,000,000	1,000,000	3	21	31	18	0

4.3.3 CIC Decimation Filter Design

As stated in the previous section, the ADC sampling rate is 125 kHz per channel. The primary benefit of such a high sampling rate is that it eases the roll-off requirements of the analog anti-aliasing filters. Less analog filter stages are needed since a larger range of frequencies can be attenuated using digital filtering techniques. However, this does demand that a large rate change take place. The data must be down-sampled to match the FCS update rate of 200 Hz. If this were to be accomplished using a traditional finite impulse response (FIR) filter, it would require over 2500 filter taps to achieve adequate stopband attenuation. Implementing such a filter is problematic due to the storage requirements and the large number of multiplication and addition operations that must be performed. It would not be possible to achieve in software due to the processing constraints. A hardware implementation would also be prohibitive because of the number of FPGA resources it would consume. Techniques such as

multi-rate filtering can be used to reduce the processing requirements, but they introduce significantly more delay into the filtered signal.

The solution was to design a custom hardware-based cascaded integrator-comb (CIC) decimation filter. CIC filters are a class of multiplier-free filters that are capable of handling large rate changes. They do not require any storage for filter coefficients and only require two's complement addition and subtraction operations. This makes it possible to implement a CIC filter much more efficiently in hardware than a typical FIR filter.

A CIC decimator consists of an integrator section, followed by a rate change, followed by a comb section. The integrator section consists of N cascaded accumulators operating at the input sampling rate, f_s. The comb section consists of N cascaded delayed difference stages, each with a differential delay of M. The comb section operates at a sampling rate of f_s/R, where R is the down-sampling factor. An N-stage CIC decimation filter will have an identical frequency response to that of N cascaded moving average filters, each with $R \cdot M$ averaged samples [61, 62]. The basic structure of a CIC decimator is shown in Figure 4.7.

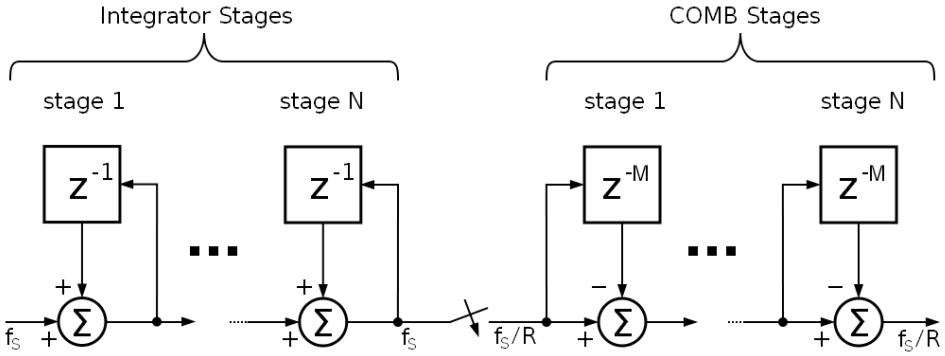

Figure 4.7: Structure of Standard CIC Decimator

The down-sampling factor, R, was fixed at a value of 625 due to the chosen input sample rate (125 kHz) and the desired FCS update rate (200 Hz). This meant that the filter response was determined by the number of cascaded stages, N, and the differential delay, M. The Filter Design Toolbox of MATLAB was used to test various CIC filter configurations and determine the

necessary parameters. It was found that a differential delay of 1 would result in insufficient attenuation past the new Nyquist frequency of 100 Hz. Increasing the differential delay to 2 moved the first frequency null to 100 Hz and resulted in improved stopband attenuation. Combined with a filter order of 3, a minimum of 40 dB of stopband attenuation was achieved. This configuration proved to be the best compromise between filter response and group delay. The frequency response of this filter is shown in Figure 4.8.

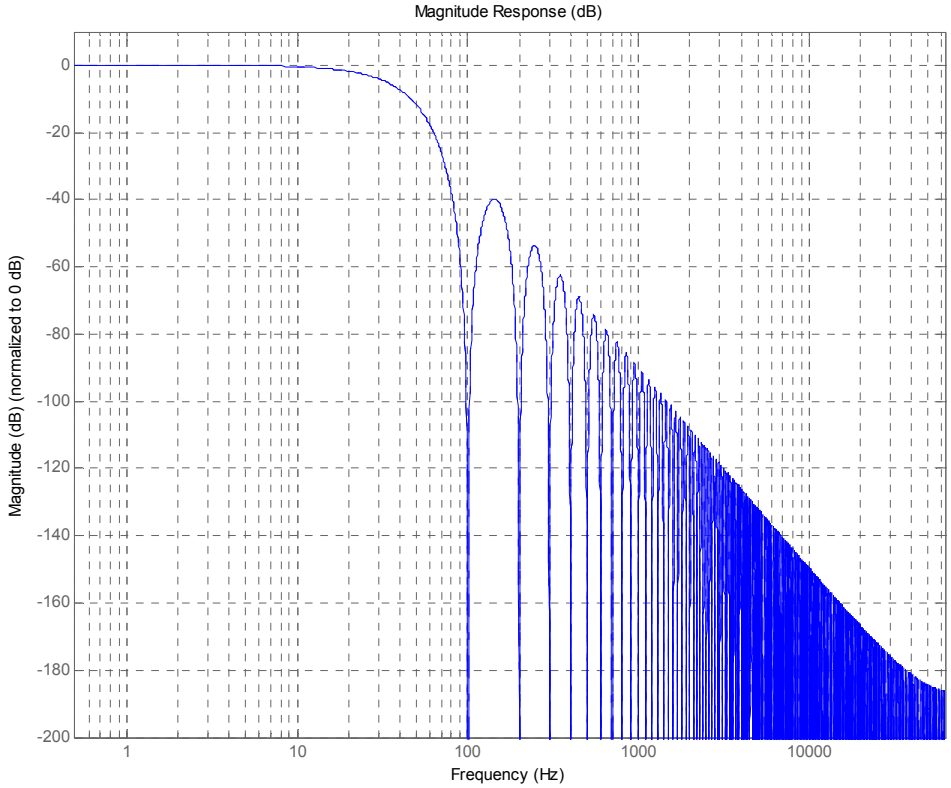

Figure 4.8: CIC Decimation Filter Frequency Response (R=625, N=3, M=2)

The gain of a CIC decimation filter is: $G = (RM)^N$. The number of bits needed to store the filter output, given an input sample width B_{in}, is: $B_{out} = \text{ceil}[N \log_2(RM) + B_{in}]$. This is also the minimum number of bits required at all stages of the filter in order to avoid data loss [61, 62]. With the chosen filter parameters, this results in a gain of 1953125000. Since the input samples are 16-bit, the integrator and comb widths must be a minimum of 47-bits.

4.3.4 CIC Decimation Filter Implementation

Although CIC filters are more economical relative to FIR filters, a straight-forward implementation of this filter would still require a fairly substantial amount of FPGA logic. Specifically, three 47-bit adders, three 47-bit subtractors, and at least nine 47-bit registers would be needed for each channel. This would require approximately 50% of the available flip-flops in the Spartan-3 FPGA, which would not be feasible. However, substantial savings can be achieved by taking advantage of hardware sharing techniques. Because the samples for the 8 input channels arrive in a time-multiplexed fashion, rather than all at once, it is possible to use one set of filtering hardware for all 8 channels. Also, because there are 60 clock cycles between input samples, it is possible to further optimize the system by reusing a single add/subtract module for all integrator and comb stages. The custom CIC filter designed for this thesis takes advantage of both optimizations.

The major components of the CIC filter include a shared add/subtract module, an associated temporary working register, an integrator & comb memory bank, and an FSM-based control module. A simplified logic diagram of the CIC filter is shown in Figure 4.9 below. In this figure, red lines indicate control signals and blue lines indicate data signals.

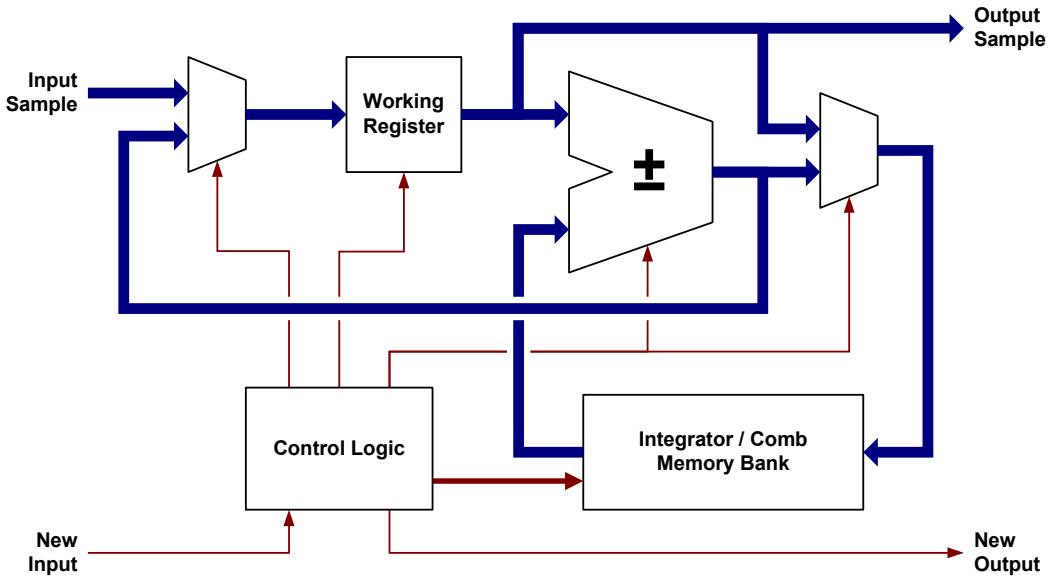

Figure 4.9: CIC Decimation Filter Logic Diagram

The working register is used to store the intermediate results calculated during each stage of the CIC filter. It is also used for buffering the input sample at the beginning of the filter process and for storing the final output sample at the end. In a standard CIC filter, this would correspond to pipelining registers inserted between each filter stage.

The integrator & comb memory is used to maintain the internal state of each integrator and comb stage. In a standard CIC filter, this would correspond to the delay elements used in the integrator feedback and comb feed-forward loops. This memory unit is implemented using Block RAM (BRAM) resources in the Spartan-3 FPGA [63]. This reduces the number of FPGA flip-flops and look-up tables for an 8-channel filter by over an order of magnitude. The dual-ported nature of BRAM is exploited in such a way that the output of the current stage can be stored at the same time that the input to next stage is being loaded. This reduces the total number of clock cycles required to process each input sample.

The control logic consists of a finite-state machine and several binary up-counters for controlling the BRAM read and write addresses, rate conversion and FSM state transfers. The FSM itself consists of three states: IDLE, SIGMA, and DELTA. The FSM stays in the IDLE

state until an input sample arrives. Once this happens, the input is stored in the working register and the FSM transitions to the SIGMA state. The memory address counter also begins sequentially iterating over the BRAM entries for the integrator stages of the current channel. During the SIGMA state, the add/subtract module is used to update the integrator values. The FSM stays in this state for *N* clock cycles and then transitions to the DELTA state. The memory address counter continues incrementing; covering the BRAM entries for the comb stages of the current channel. During the DELTA state, the add/subtract module is used to update the appropriate comb stages if enabled by the rate conversion counter. This happens once for every *R* inputs of each channel and results in a new output sample at the end of the state. The FSM stays in this state for *N·M* clock cycles, even if the comb stages are not enabled. This is done to simplify the address counter and FSM logic, reducing FPGA resource requirements. The total number of clock cycles required per input is *N·(M+1)+1*.

A diagram illustrating this process for a filter with N=2 and M=1 is provided below in Figure 4.10. This diagram shows both the state transitions and the data flow during each cycle of the filtering process. Although not shown, a write enable signal for the BRAM and working register is used to control when the comb stages are active.

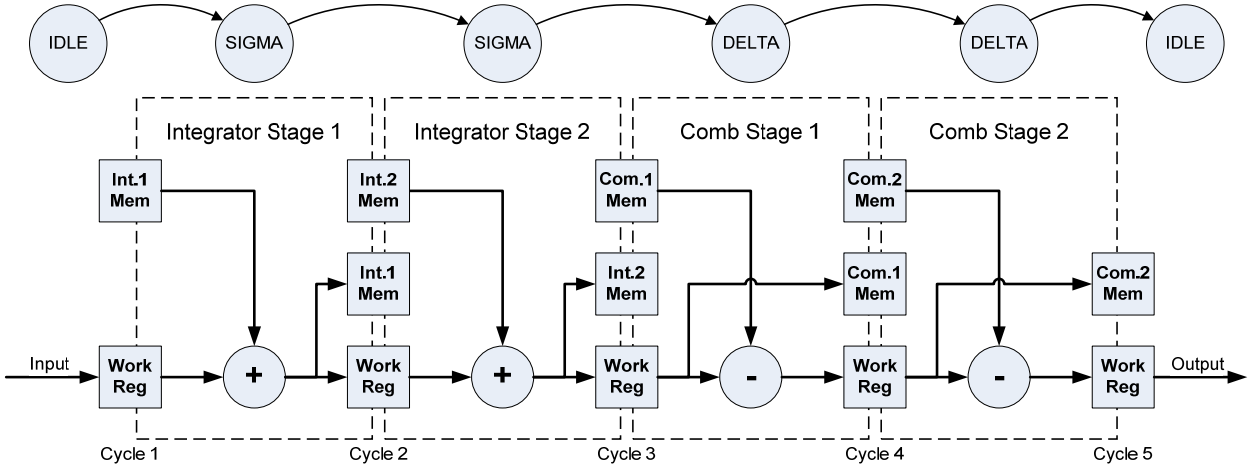

Figure 4.10: CIC Decimation Filter Process

The VHDL code for this filter is fully parameterized. The number of channels (*C*), number of filter stages (*N*), decimation factor (*R*), and differential delay (*M*) are all configurable. As mentioned previously, the chosen parameters are: C=8, N=3, R=625, M=2. This results in a usage of 103 slices, about 2.9% of the total available in the ICM Spartan-3. However, the optimized nature of this implementation combined with the usage of BRAM means that the number of channels and number of filter stages could be increased without significantly increasing the required FPGA resources. There is no artificial limit to the number of channels or stages. In practice, this is only limited by the desired sample rate and the amount of BRAM that can be dedicated to the core. With the above filter configuration, this core can support up to 56 channels while only requiring 2 BRAMs. It can support up to 113 channels if 3 BRAMs are used. In both cases, the number of slices does not noticeably change. The resource usages for various configurations are shown below in Table 4.5. The chosen configuration is highlighted.

Table 4.5: CIC Decimation Filter FPGA Resources

Filter Config				Spartan-3 Resources			
C	N	R	M	Slices	Flip-Flops	LUTs	BRAM
8	3	625	2	103	82	191	2
16	3	625	2	103	84	191	2
32	3	625	2	105	86	194	2
56	3	625	2	105	86	194	2
113	3	625	2	103	88	192	3
32	4	625	2	118	98	217	2
32	5	625	2	140	108	257	2

It should be noted that Xilinx provides a configurable CIC filter, the CIC Compiler v1.2, with its CORE Generator software. The Xilinx CIC Compiler v1.2 was evaluated prior to developing the custom CIC filter discussed above, but was ultimately not used due to its relatively high FPGA resource requirements. With the same parameters ultimately chosen for the custom core (C=8, N=3, R=625, M=2), the Xilinx core requires 934 slices. This is over 25% of the total resources in the Spartan-3. Although this could be made to fit in the ICM design, it

would leave few resources available for future use. If the number of channels is increased to 16, in order to accommodate additional analog sensors, the Xilinx core would require 1869 slices. This would make it impossible to fit in the current ICM design. The Xilinx core also does not support unsigned data, output data widths greater than 48-bits, or more than 16 channels. The custom CIC decimation filter designed by the author has none of these limitations and uses significantly less FPGA logic.

4.3.5 Unsigned Binary Divider

The filtered samples must be stored in an intermediate memory buffer before the CPU can access them via the local bus. The large gain and resulting register growth of the CIC decimation filter mean that additional FPGA resources would be required to construct this buffer if the filtered samples were stored in their entirety. Compared to the 16-bit ADC samples, the 47-bit filtered samples would require almost three times the memory. Also, because the CPU and local bus are only 32-bit, additional clock cycles would be required to access and manipulate the samples. To prevent these issues, a gain normalization stage was added to reduce the width of the filtered samples to a more usable size. It was decided that the filtered samples should be restored to their original width of 16 bits, since this offers more than enough resolution for the analog sensors used in this project.

This was accomplished by creating a simple unsigned divide-by-constant module to divide each filtered output sample by the gain of the CIC filter. This module is fully configurable; VHDL generics are used to specify the divisor constant, dividend width and maximum dividend input. In practice, these parameters are automatically calculated by the analog capture core based on the specified filter configuration. The divider's quotient width and

required number of clock cycles are also calculated during synthesis. Synthesis time optimizations are used to minimize the total amount of logic resources consumed.

In cases where the divisor happens to be a power of two, binary division is performed by simply discarding the appropriate number of least significant bits from the dividend. This requires no logic and produces the quotient in zero clock cycles.

In all other cases, the quotient is computed using a fairly conventional iterative shift and subtract method. Any powers of two are still factored out of the divisor constant and the dividend input is truncated appropriately. This can reduce the number of bits in the working register, leading to a slight decrease in needed FPGA resources. The number of clock cycles required for this implementation is equal to the number of bits in the quotient plus one. The first clock cycle is used to load the divided into the working register and initialize the division process. Each subsequent clock cycle produces a single bit of the quotient.

The specified divider parameters and calculated constants are shown below in Table 4.6. The FPGA resources required for this divider are shown below in Table 4.7. The number of slices utilized is approximately 1.6% of the total slices available in the ICM Spartan-3.

Table 4.6: Unsigned Divider Specifications

Divisor Constant	1,953,125,000
Dividend Width	47-bit
Dividend Max	127,998,046,875,000
Quotient Width	16-bit
Quotient Max	65,535
Clock Cycles	17

Table 4.7: Unsigned Divider FPGA Resources

Spartan-3 Resources			
Slices	**Flip-Flops**	**LUTs**	**BRAM**
58	50	96	0

It should be noted that the resolution enhancement method previously discussed in section 3.5.3.1 could be accomplished by simply right shifting the calculated divisor constant by the appropriate number of bits. This would of course have an effect on the logic utilization of the divider.

4.3.6 Sample Memory & Bus Interface

The memory used as storage for the filtered samples is implemented in a double-buffered configuration. Filtered samples from the divider module are written to one half of the memory, while the other half is connected to the local bus interface. Every time a complete set of samples is written to the memory, the two buffers are swapped and an interrupt is generated. The CPU can then read the new set of samples using memory mapped I/O. This double buffering also acts as a form of synchronization between the system clock domain and the analog capture core clock domain.

The double buffered memory is realized using distributed RAM in the Spartan-3 FPGA. Distributed RAM is made up of one or more LUTs, each providing a 16-deep x 1-bit synchronous memory element [64]. In this case, a total of 32 LUTs are used as dual-ported memory in a 16 x 16 bit arrangement.

Versions of this Analog Capture Core were created for interfacing to either the OPB or PLB, with the only difference being the wrapper logic needed for the chosen peripheral bus. The total FPGA resources required for both versions of this core are presented in Table 4.8 below. In the ICM Spartan-3, this core interfaces with MicroBlaze via the OPB bus.

Table 4.8: Analog Capture Core FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
Analog Capture Core + OPB Interface	219	173	409	2
Analog Capture Core + PLB Interface	228	232	425	2

4.4 Module-to-Module Link

A simple full-duplex bus was created to allow communication between the ICM and FCM FPGAs. This bus, known as the Module-to-Module (M2M) Link, achieves very low latency operation while requiring minimal FPGA resources. Specifically, it can be implemented with less than 100 Spartan-3 slices and requires no external hardware. This is possible due to the close proximity of the two FPGA modules, which simplifies the requirements for high-speed, error-free communication. In contrast, the Xilinx IP cores implementing standard high-speed interfaces, such as Ethernet or USB, need an order of magnitude more FPGA resources. These interface protocols are designed to allow communication over longer distances and require external PHY hardware to handle the physical layer.

The M2M Link itself consists of a pair of 8-bit unidirectional data channels, one for data transfer in each direction. Each channel functions independently and has its own clock signal and set of control signals. The transmitter ↔ receiver interface for an M2M channel is shown in Figure 4.11. The individual signals are described in Table 4.9.

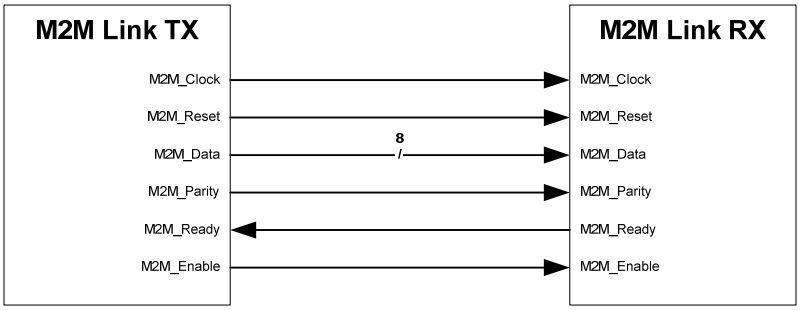

Figure 4.11: Module-to-Module Link Interface Diagram

Table 4.9: Module-to-Module Link Signals

Signal	Description
M2M_Clock	Source-Synchronous Clock - Rising Edge Sensitive
M2M_Reset	System Reset Signal - Active High
M2M_Ready	Receiver Ready Signal - Active High
M2M_Enable	Write Enable Signal - Active High
M2M_Data	Data Value - 8-bit Bus
M2M_Parity	Parity Bit - Even Parity

The clock signal is source-synchronous and drives both the transmitter and receiver logic. Clock domain synchronization occurs at the receiving system. The reset and ready signals are used to ensure that both sides are properly initialized after system power-up and FPGA configuration. The ready signal is also used along with the enable signal to control the data transfer process. The parity signal is optional and can be used as a simple means of error detection.

Both the transmitter and receiver logic have 2kB data buffers, limiting individual transfers to a maximum of 2kB each. During transfers, a single byte will be copied every clock tick. This means that each transfer will require a maximum of 40.96 microseconds when operating at 50 MHz. It should be noted that, in practice, significantly less than 2kB of data will be transferred between the ICM and FCM for each FCS update cycle.

The M2M Link, like many packet-based interfaces, will generate a single receive interrupt for each block of data transferred. This is in contrast to common sensor interfaces, such as SPI and RS-232, which typically generate an interrupt for every byte received. For instance, a serial UART operating at 115200 baud can easily generate over 10,000 interrupts per second. Because the ICM handles these sensor interfaces, the FCM will be isolated from the resulting interrupts and the FCS software can operate much more efficiently. In the case of the FCM, the M2M receiver will be the primary interrupt source, with interrupts only occurring at the beginning of each FCS update cycle.

4.4.1 M2M Transmit Core

The major components of the M2M Transmit Core are the 2kB data buffer, status and control registers, address counter, and FSM control logic. A logic diagram depicting these

components is shown below in Figure 4.12. All logic runs directly off of the local bus clock. The M2M_Clock and M2M_Reset signals are aliases for their local bus equivalents.

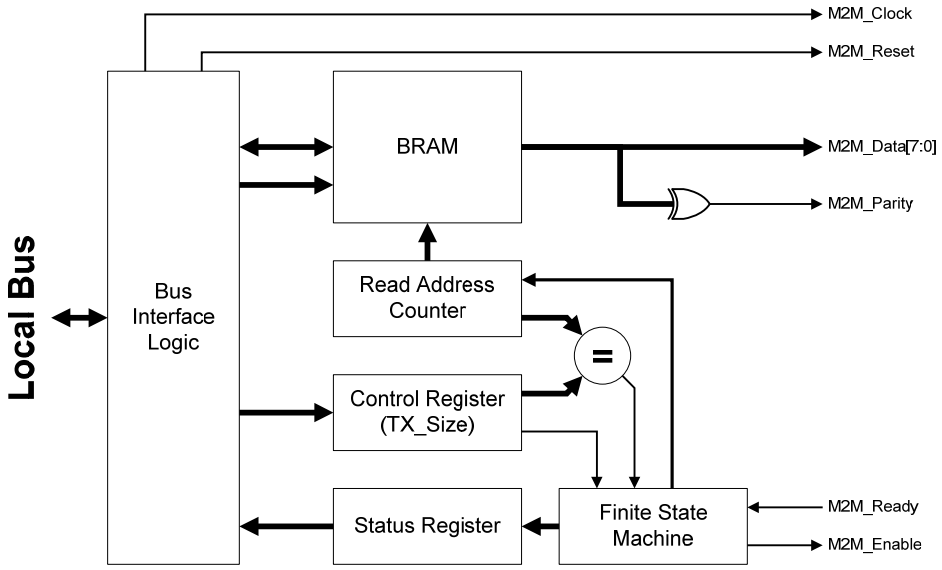

Figure 4.12: Module-to-Module Transmit Logic Diagram

A single dual-ported BRAM block is used to implement the 2kB data buffer. This data buffer is mapped to the lower 2kB of the peripheral core's address space and can be accessed while the transmitter is idle. The Xilinx BRAMs have the capability of having different data and address widths for each port [63]. This ability is utilized to simplify the data interfaces and reduce logic usage. Port A of the BRAM is configured as an 8-bit read-only port and is connected to the M2M data bus. Port B of the BRAM is configured as a 32-bit read-write port and is connected to the local bus interface.

The status and control registers are mapped to the upper 2kB of the peripheral core's address space. A read operation in this memory range will access the status register, while a write operation will access the control register. The status register indicates when the local transmitter logic is ready, while the control register is used to initiate the transfer. Writing a value, N, to the control register will result in the lower N bytes of BRAM data being copied across the M2M Link.

The actual transfer process is controlled by the FSM and read address counter. The FSM consists of three states: TX_Idle, TX_WaitReady, and TX_Sending. The initial state is TX_Idle, and the FSM remains in this state until the user software writes to the control register. Once this happens, the FSM transitions to TX_WaitReady and monitors the M2M_Ready signal to determine when the receiver is ready. When the receiver is ready, the FSM enters the TX_Sending state and begins sequentially transmitting the BRAM contents by controlling the read address counter and M2M_Enable signal. Once the read address is equal to the control register value, all data has been transferred and the FSM returns to the TX_Idle state. A state diagram illustrating this process is shown in Figure 4.13.

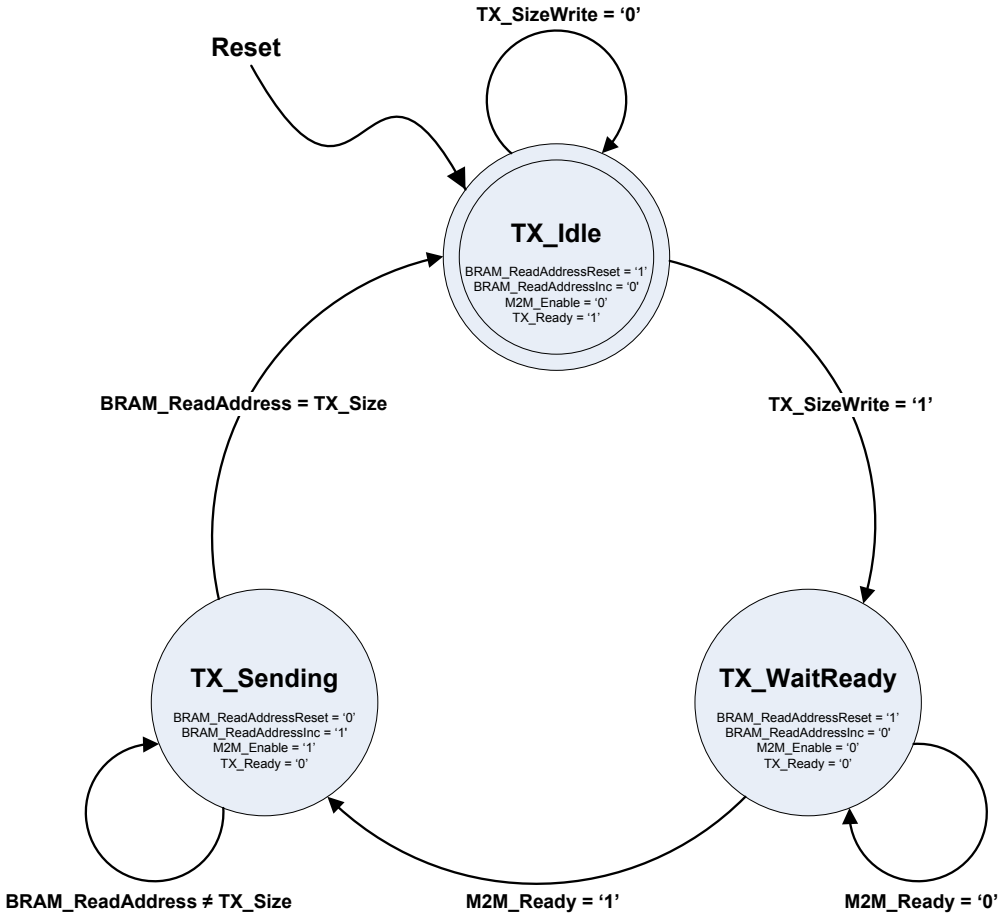

Figure 4.13: Module-to-Module Transmit Finite State Machine

The FPGA resources required for both the OPB and PLB versions of this core are shown below in Table 4.10. The additional overhead of the PLB interface is particularly noticeable in this case, more than doubling the total number of slices needed.

Table 4.10: M2M Transmit FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
M2M Transmit + OPB	41	31	71	1
M2M Transmit + PLB	103	161	82	1

4.4.2 M2M Receive Core

The M2M Receive Core consists of a 2kB data buffer, parity error register, address counter, synchronization logic, and FSM control logic. A logic diagram depicting these components is shown below in Figure 4.14. The FSM and address counter run off of the M2M_Clock. The parity error register runs off of the local bus clock. The data buffer and synchronization logic operate in both clock domains.

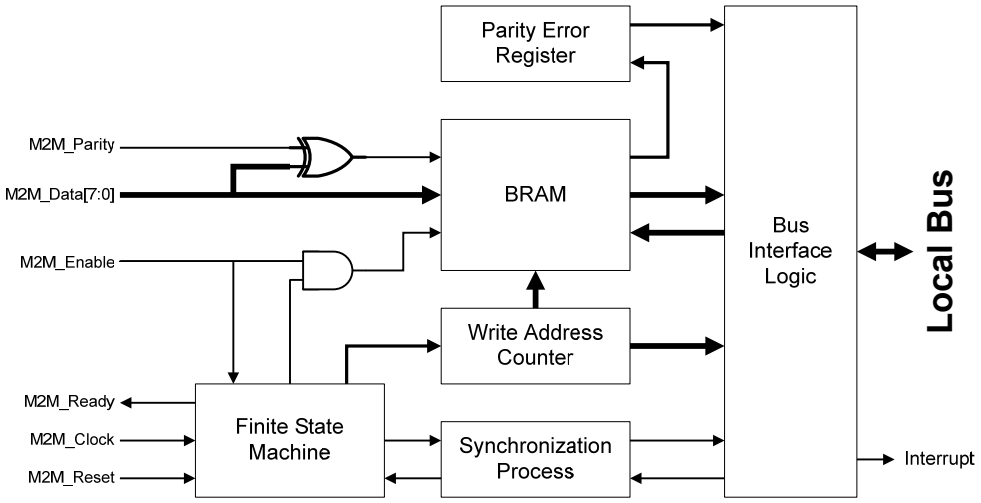

Figure 4.14: Module-to-Module Receive Logic Diagram

Like in the M2M Transmit Core, the data buffer is implemented using a dual-ported BRAM block. This buffer holds both the received data and the corresponding parity error information. Port A is configured as a 9-bit write-only port and connected to the M2M_Data

signal (8-bit) and parity error signal (1-bit). Port B is configured as a 36-bit read-only port and connected to the local bus interface (32-bit) and parity error register (4-bit).

The BRAM buffer is mapped to the lower 2kB of the peripheral core's address space. The write address counter is mapped to memory offset 2048 and the parity error register is mapped to memory offset 2052. After a transfer is complete, the write address counter can be read to determine the amount of received data in the BRAM buffer. The parity error register can be used to determine if any parity errors are associated with the last 32-bit word read from the BRAM.

The synchronization logic is responsible for transferring the local bus reset signal into the M2M_Clock domain. It also transfers the RX_Done signal into the local bus clock domain. This signal indicates when an M2M transfer has completed and controls read access to the BRAM buffer, address counter, and parity error register. The RX_Done signal is also used to generate the M2M Receive interrupt signal. The user software can acknowledge a transfer by performing a write operation within the peripheral core's address space.

The receive process is controlled by the FSM and write address counter. The FSM consists of four states: RX_Reset, RX_Idle, RX_Transfer, and RX_WaitUser. The initial state is RX_Reset and the FSM will enter this state if a reset occurs on either the M2M bus of the local bus. If a reset occurs during a transfer, the FSM will stay in this state until the transfer completes to prevent it from presenting incomplete data to the user software. After reset, the FSM enters the RX_Idle state and monitors the M2M_Enable signal. A transfer begins when M2M_Enable goes high and the FSM enters the RX_Transfer state. During this state, the write address is incremented and data is sequentially written to the BRAM. After the transfer is complete, the FSM enters the RX_WaitUser state and waits for the user software to read the data and

acknowledge the transfer. The FSM then returns to the RX_Idle state. A state diagram illustrating this process is shown in Figure 4.15.

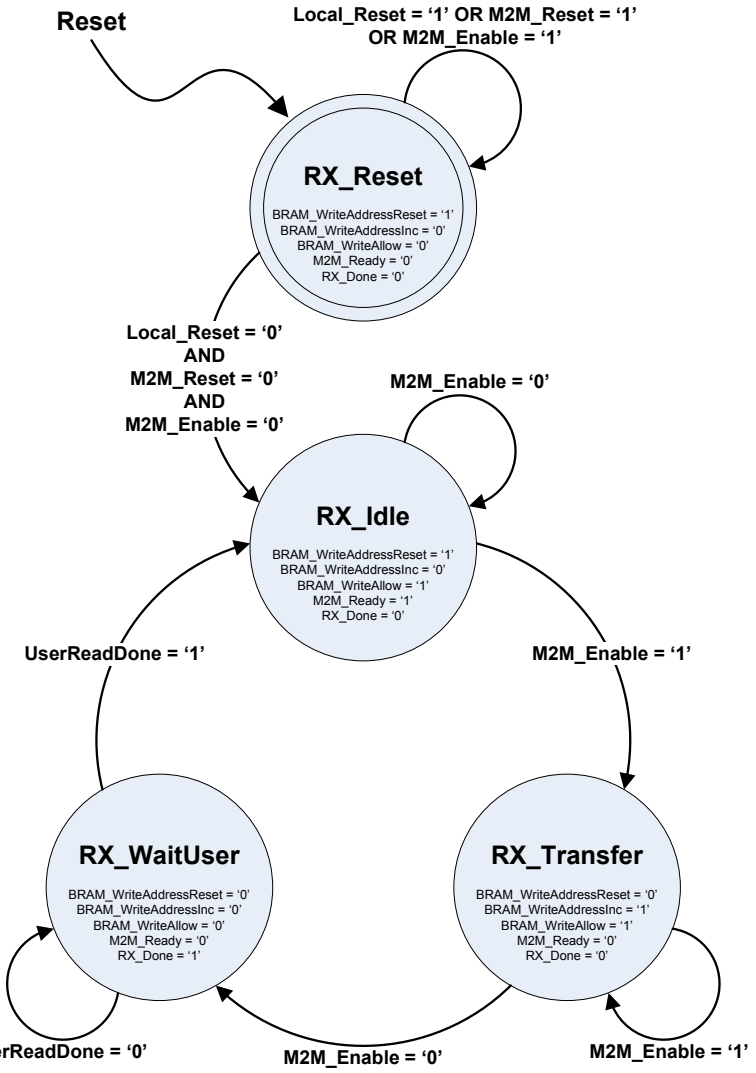

Figure 4.15: Module-to-Module Receive Finite State Machine

The FPGA resources required for both the OPB and PLB versions of this core are shown below in Table 4.11.

Table 4.11: M2M Receive FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
M2M Receive + OPB	50	29	88	1
M2M Receive + PLB	90	120	98	1

4.5 PWM Write Core

A custom pulse width modulation (PWM) output core was created. This core is responsible for generating the control signals needed to drive the aircraft servos during autonomous operation. Each control signal consists of a series of pulses, with the duration of the pulse corresponding to the commanded servo position. Most RC servos accept pulses in the range of 1 to 2 ms, with 1.5 ms corresponding to the center position.

A logic diagram of this core is shown below in Figure 4.16. The periodic timer along with the pulse width registers and their associated less-than comparators are the primary components responsible for generating the PWM pulse trains. The periodic timer is a free-running counter that continuously increments and then automatically resets after reaching its maximum value. Each pulse width register holds a value describing the duration of the pulse to be generated. An output signal will be high whenever the timer value is less than the value in the corresponding pulse width register, and low otherwise.

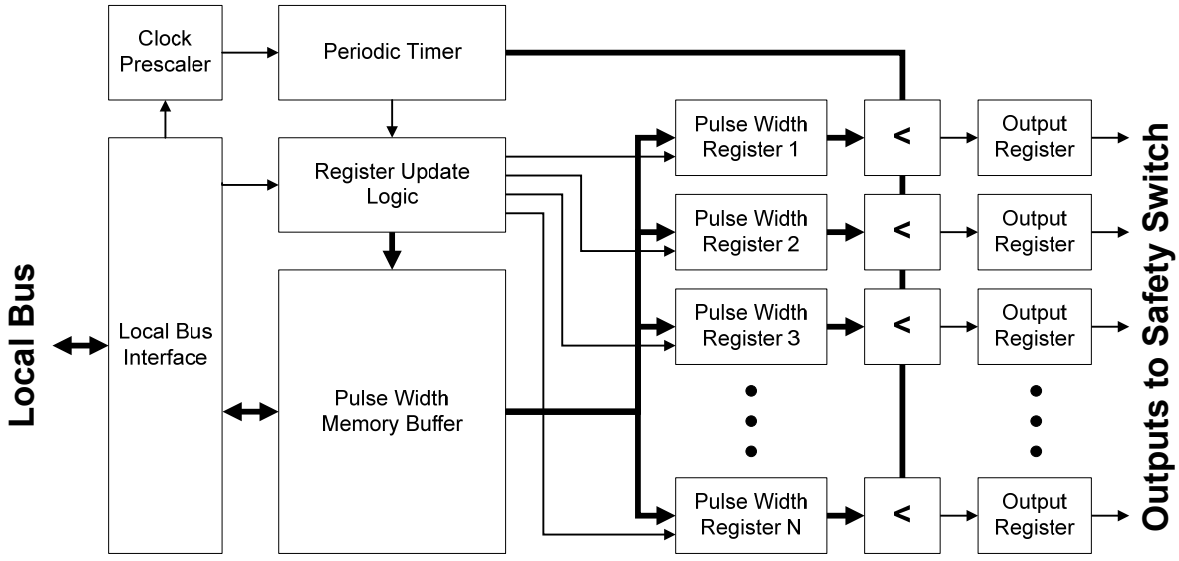

Figure 4.16: PWM Write Logic Diagram

The clock prescaler is used to control how often the periodic timer is incremented by dividing the frequency of the main system clock. This effectively determines the resolution of

the PWM output signals. The pulse width memory buffer is constructed using distributed RAM and is used to double buffer the pulse width values. This is necessary to prevent glitches in the output control signals. Such glitching could occur if the pulse width register value was changed from a value less than the timer to a value greater than the timer. To prevent this, the registers are only updated at the end of each timer period.

This core is fully parameterized using VHDL generics. The number of channels, the prescaler value, the register width and the timer period are all configurable. The parameters chosen for this design are shown in Table 4.12. A prescaler value of 50 was chosen, giving a pulse resolution of 1 microsecond due to the 50 MHz system clock. The register widths were set to 12-bit, allowing for a maximum pulse width of 4.095 ms.

Table 4.12: PWM Write Configuration

Clock Prescaler	50
PWM Register Width	12
Number of Channels	8
Pulse Period	20000

The update period was set to 20 ms for safety reasons. This is because many analog servos can only tolerate update rates of 50 Hz maximum. Updating these servos too fast can result in permanent damage to the servo. Newer digital servos do not have this limitation and can be updated at rates of over 300 Hz [65]. If this system is used with digital servos, the pulse period can be decreased to allow for quicker update rates.

The FPGA resources required for both the OPB and PLB versions of this core are shown below in Table 4.13.

Table 4.13: PWM Write FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
PWM Write + OPB	151	136	199	0
PWM Write + PLB	200	214	217	0

4.6 PWM Read Core

A custom PWM input core was also created. This core is responsible for reading the manual control signals from the RC receiver. These manual control signals can be used by the FCS to implement various semi-autonomous flight modes. This core is also responsible for monitoring the manual/auto control channel and generating the mode select signal for the onboard safety switch hardware.

In addition, this core can optionally be configured to monitor the throttle channel. Some RC receivers use this channel to indicate a loss of link by setting the PWM signal to a fail-safe value. This condition can be detected and an error signal can be generated. This can potentially be used by the autopilot to enter a "return home" mode.

This core essentially performs the opposite function of the PWM Write Core. Pulse width counters are used to measure the length of time that the PWM signals stay high. Whenever a signal transitions from low to high, the corresponding counter begins incrementing. When the signal transitions from high to low, the counter value is transferred into the corresponding pulse width register.

The pulse width registers are connected to the local bus and can be read by the user software. These registers are also used by the mode select and fail-safe detection logic. The appropriate register values are compared with predetermined values to generate the mode select and fail-safe signals.

An optional timeout detection counter can be enabled. If an input signal remains inactive for two consecutive periods of the counter, a timeout condition occurs. When this happens, the corresponding pulse width register is reset to zero to indicate that no PWM signal is detected.

A logic diagram of this core is shown below in Figure 4.17.

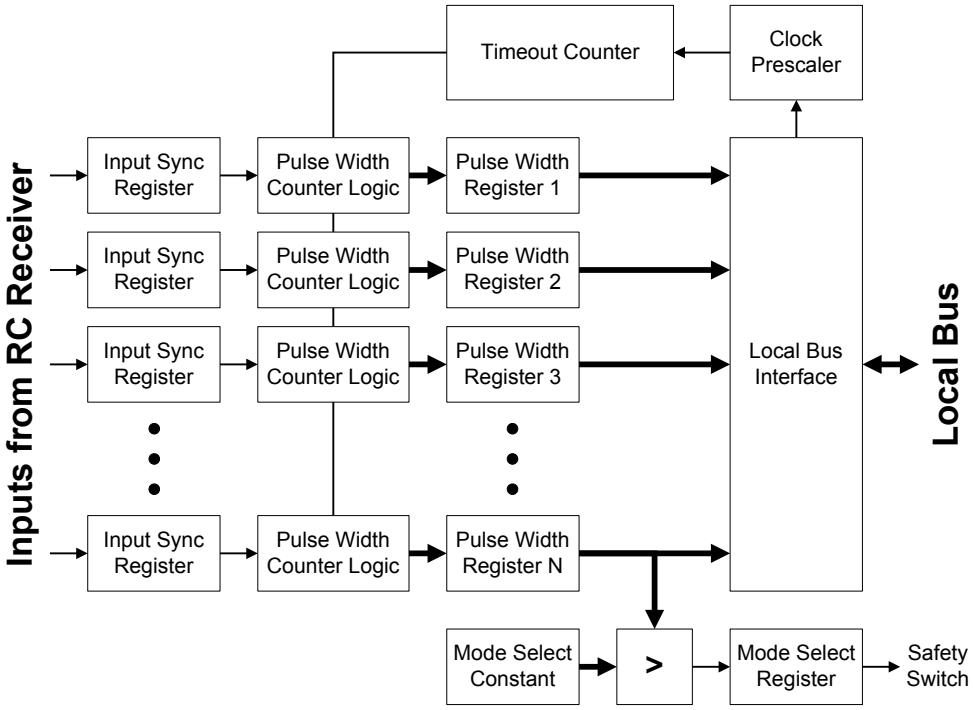

Figure 4.17: PWM Read Logic Diagram

The configuration parameters chosen are shown in Table 4.14. The prescaler value and register widths are set the same as in the PWM Write Core. The mode select value is set to 1500, meaning that the control mode threshold is 1.5 ms. Pulses shorter than this will put the system in manual control mode, while longer pulses will enable autonomous control.

Table 4.14: PWM Read Configuration

Clock Prescaler	50
PWM Register Width	12
Number of Channels	9
Timeout Interval	32768
Mode Select Channel	9
Mode Select Value	1500
Fail Safe Channel	3
Fail Safe Value	950

The FPGA resources required for both versions of this core are shown in Table 4.15.

Table 4.15: PWM Read FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
PWM Read + OPB	257	289	378	0
PWM Read + PLB	301	357	395	0

4.7 Simple FSL Cores

In addition to the more complex memory mapped IP cores discussed above, several simple FSL-based cores were also created. These cores handle very basic functions and require few FPGA resources. The direct FSL bus was chosen due to the simple nature of these cores. This bus essentially adds zero overhead, due to its point-to-point nature and the lack of memory addressing. In comparison, the OPB and PLB bus interfaces would actually require more logic resources than the cores themselves. The cores created include counters for measuring CPU cycles and microseconds, an LED driver module and a general purpose I/O core. The FPGA resources for all four cores are shown below in Table 4.16.

Table 4.16: Simple FSL Core FPGA Resources

	Spartan-3 Resources			
	Slices	**Flip-Flops**	**LUTs**	**BRAM**
CPU Cycle Counter	17	32	33	0
Microsecond Counter	23	39	44	0
Status LED Driver	18	22	24	0
Config Jumpers GPIO	0	2	0	0

The CPU cycle counter is a resettable 32-bit register that is incremented every clock cycle. This core is primarily used for code profiling and local system timing analysis, among other things. The guaranteed single cycle latency of the direct FSL bus allows for cycle accurate measurements.

The microsecond counter is a 32-bit register that is incremented once every microsecond (50 clock cycles at 50 MHz). A similar microsecond counter is also utilized in the FCM FPGA and runs off of the same clock as the ICM microsecond counter. The two counters are synchronized using the M2M clock and reset signals from both FPGAs. Lab testing confirms that both counters stay synchronized to within 0.5 microseconds and do not drift apart over time. These counters measure time since system power-on, and provide timestamping for all data

measured, logged, and transferred between the two FPGAs. This allows for accurate timing of all phases of the sensor → processing → actuator feedback loop.

The LED driver module is used to update the ICM status & debug LEDs. These LEDs are connected to an LED current sink IC that is controlled using a 3-wire serial bus. This core is essentially a write-only SPI interface that converts the 8-bit LED value to serial and instructs the IC to update the LEDs.

The general purpose I/O core is simply a register that is connected to several I/O buffers in the Spartan-3 FPGA. It allows direct transfer of parallel data between the MicroBlaze and FPGA pins. It is currently used to read configuration jumpers, but could potentially be used for other purposes such as payload control.

Chapter 5: ICM Software

The software implemented in the Instrumentation Control Module will be covered in this chapter. This will include a discussion of the software's control flow and various sub-tasks along with the protocols utilized to communicate with the FCS and ground control station (GCS). Information will also be provided on the generic sensor interface created as well as details of specific sensor device drivers. Note that the use of the term FCS in this chapter will refer to the autopilot software running on the FCM.

5.1 Software Overview

The ICM software is primarily responsible for taking sensor measurements and converting them to a format usable by the FCS. This includes parsing serial data provided by digital sensors, processing analog sensor readings, and performing additional digital filtering functions. The ICM software also handles configuration parameters and commands from the GCS and relays FCS telemetry data back down to the GCS. Actuator and payload commands are received from the FCS and output to the appropriate control interface.

The Xilinx EDK 10.1i software suite was used to develop and test the ICM software. It contains a collection of tools and libraries that facilitate creation of MicroBlaze applications. User software is written in C and compiled using MB-GCC, a MicroBlaze specific version of the GNU Compiler Collection (GCC) based on release 4.1.1. Standard C library functions are provided by a customized version of Newlib and statically linked at compile time. Basic low-level drivers are also provided for interfacing with any Xilinx IP Cores used.

5.2 Main Control Loop

The ICM software itself consists of a continuously running main loop, shown in Figure 5.1 below, and several interrupt service routines for low-level I/O handling (not shown). Additional details are provided in callouts, with starred lines indicating optional events that depend on the previous un-starred event.

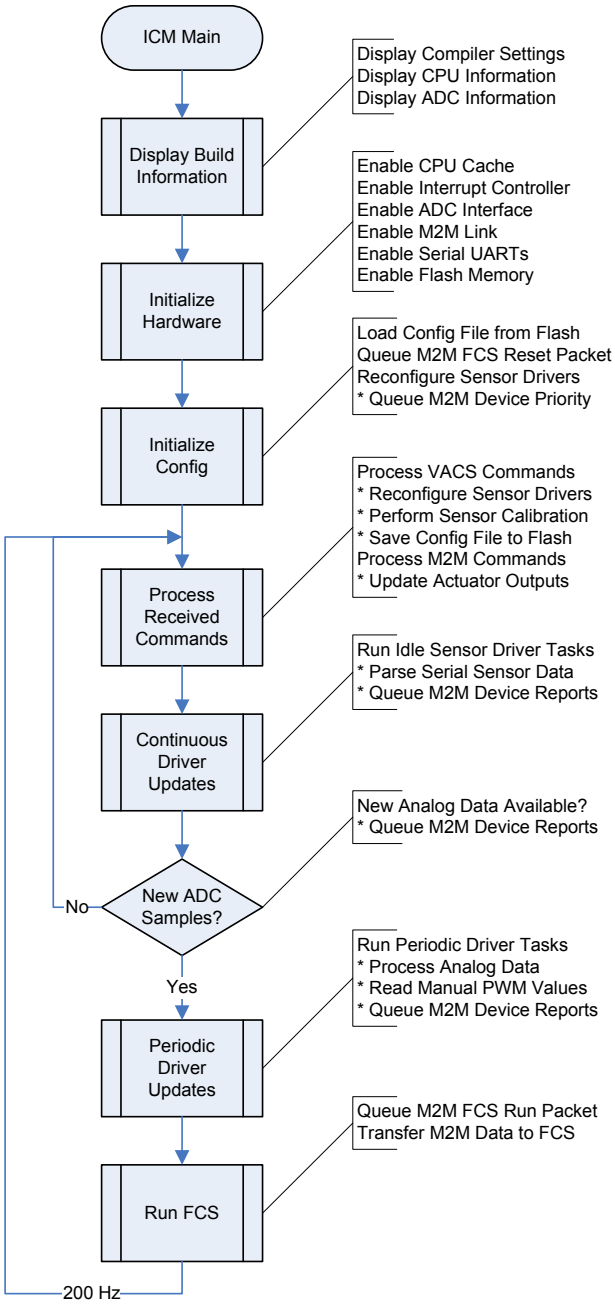

Figure 5.1: Top-Level Diagram of ICM Software Main Loop

At startup, the main program is first copied from flash memory into SRAM, verified, and then executed. Following main program load, the system hardware and low-level device drivers are initialized and interrupts are enabled. The configuration file is then loaded from flash memory and applied, reconfiguring the sensor drivers. A reset command is also sent to the FCS via the M2M bus, along with default sensor priority information.

If enabled, detailed debugging information will be printed to a dedicated serial console during startup. This information includes build settings such as compiler flags and hardware parameters. The result of each device initialization step and the status of the configuration memory will also be output. This information is intended to aid in development and troubleshooting, but can be disabled to save program memory. An example of the information printed at startup is shown below in Figure 5.2.

Figure 5.2: ICM Startup Messages

Following the startup sequence, the program will enter the main control loop. This loop consists of two stages, a continuous portion and a periodic portion operating at 200 Hz. The continuous portion handles idle tasks and listens for events that can occur independently of the ICM. This includes GCS commands received via the radio modem and FCS commands received through the M2M bus. Data from serial based sensors is also received and processed throughout this phase. Once new data is available from the analog capture core, which occurs every 5ms, the main loop will enter the periodic segment. During this stage, manual control signals from the RC receiver are read in and analog sensor data is filtered and processed. During both the continuous and periodic stages, processed sensor data can be queued up for transmission across the M2M bus. At the end of the periodic stage, an "FCS Run" command will be added to the M2M queue, after which all queued data will be sent to the FCS.

Ultimately, the analog capture core in the ICM FPGA controls the timing of the ICM software's periodic update loop, which in turn drives execution of the control and navigation routines in the FCM. One consequence of this arrangement is that the FCS only needs to listen for data from the ICM during flight. After receiving the ICM data, the FCS control loop can execute largely uninterrupted, minimizing the nondeterministic effects of I/O interrupts. Upon completion, the FCM software will return updated telemetry and actuator values to the ICM via the M2M bus.

5.3 Wireless Communications

All communications between the aircraft and ground station currently utilize the VACS binary protocol. This is a simple packet format intended for point-to-point communication via a wireless serial link. It supports continuous communication between the GCS and a single UAV. Infrequent communication with two or more UAVs is possible in some cases, but reliable

transmission is not guaranteed due to lack of low-level channel control. This protocol is used in order to maintain backwards compatibility with the current GCS software.

The packet format itself consists of an eight-byte header, a variable length payload of up to 1024 bytes, and a two-byte checksum as shown in Table 5.1. All two-byte fields are stored in little-endian byte order. Payload format and byte order are implementation defined. The checksum algorithm used is a variant of the 16-bit Fletcher checksum using two's complement arithmetic. It is calculated over all bytes between sync byte B and the checksum.

Table 5.1: VACS Packet Format

Packet Field	Size
Sync Byte A (0x76)	1
Sync Byte B (0x63)	1
Destination Address	1
Source Address	1
Message ID	2
Payload Length	2
Payload Data	0 - 1024
Checksum	2

Serial parsing is implemented using a simple FSM construct with recursive resynchronization capability. After reception of a sync byte pair, the packet is assembled sequentially. If an error occurs during packet assembly, such as an invalid length or incorrect checksum, the packet buffer is not discarded. Instead, the FSM parser is restarted and the buffer is rescanned for additional sync bytes. This process is implemented recursively and can potentially execute multiple times before recovering synchronization. A maximum recursion depth of 8 is set to limit stack usage and prevent the possibility of stack overflow. This recursive packet scanning technique helps to reduce the impact of momentary data loss and decrease the number of retransmissions needed. Without rescanning, a corrupt packet resulting from dropped data could result in the rejection of not only itself, but of one or more subsequent packets as well. This is due to the fact that when operating in "Transparent Mode", the radio modems are

unaware of the serial packet structure being used. Outgoing serial data is buffered and split into smaller frames for transmission by the modem. Because these frames are uncorrelated with the VACS packets, a dropped frame can occur at any point in the VACS packet stream. Figure 5.3 shows one possible example of this occurrence.

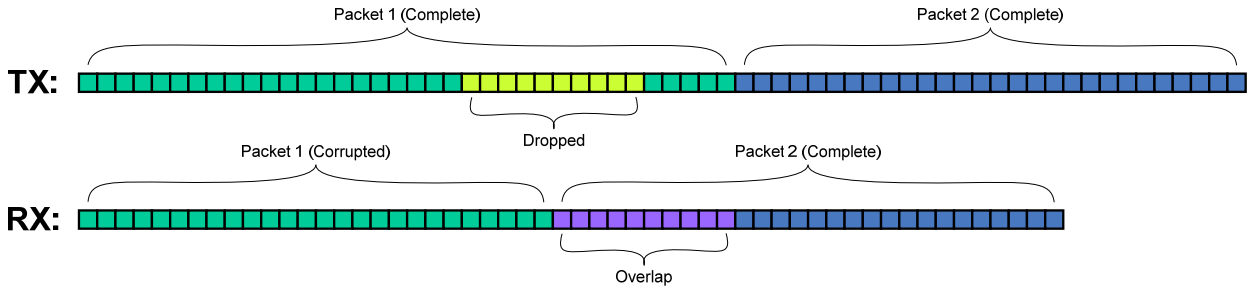

Figure 5.3: Potential Packet Loss Scenario - Recursive scanning allows recovery of packet 2.

There are four main types of VACS packets: report, command, request, and reply. Report packets are unreliable and do not require a response. They are typically used by a UAV to send telemetry data, but can also be sent from the GCS. Command and request packets are reliable and require a corresponding reply packet from the remote end-point. Lack of a reply initiates retransmission after a set timeout. These packets are currently used by the GCS to transfer configuration parameters to and from the UAV.

The ICM currently utilizes a set of reliable VACS packets to allow for dynamic sensor reconfiguration and calibration from the GCS. Any unhandled VACS packets received from the GCS will be forwarded to the FCS via the M2M bus. Telemetry data from the FCS will also be converted to VACS report packets and periodically sent to the GCS. A leaky bucket rate limiting algorithm is employed to prevent saturating the radio modem's limited bandwidth. By default, the FCS to GCS transmit rate will be limited to 80% of the modem's bandwidth. This ensures that sufficient bandwidth remains available for command and request packets to arrive from the GCS.

5.4 Inter-Module Communications

All communications between the FCM software and the ICM software utilize the M2M packet-based protocol. This protocol uses a simple binary packet format and contains both predefined and generic packet types. All M2M packets consist of a four byte header followed by a variable length payload of 0 to 2044 bytes. The first two bytes of the header specify the packet type, while the second two bytes specify the payload size. No checksum is necessary due to the reliable nature of the M2M link. When placed in the M2M BRAM buffer, all M2M packets will be word aligned to 32-bit boundaries. This improves the performance of bus transactions to and from the M2M BRAM buffers.

There are four basic classes of M2M packets, distinguished by their payload structures. A total of ten specific M2M packet types are currently implemented using the four basic payloads. Table 5.2 lists the ten packet types, while Table 5.3 through Table 5.6 show the four payload structures used. The set of Device* packets are used for transferring sensor and actuator information between the ICM software and the FCS. The VACS* packet types utilize the Comm Data payload format and are used for relaying data between the FCS and GCS. The FCSRun and FCSReset packets are used to control execution of the FCS and reset the internal FCS state, respectively.

Table 5.2: M2M Packet Types

Type	Packet Name	Payload Structure	Source	Description
0x10	DeviceReport	M2M Device Data	ICM	Device Driver Report
0x11	DeviceCommand	M2M Device Data	FCM	Device Driver Command
0x12	DevicePriority	M2M Device Priority	ICM	Device Driver Priority
0x13	DeviceQuery	M2M Basic Command	FCM	Device Driver Query
0x20	VACSCommand	M2M Comm Data	ICM	Command Message from GCS
0x21	VACSRequest	M2M Comm Data	ICM	Request Message from GCS
0x22	VACSReport	M2M Comm Data	FCM	Telemetry Message from FCS
0x23	VACSAck	M2M Comm Data	FCM	Acknowledgement from FCS
0xF0	FCSReset	M2M Basic Command	ICM	FCS State Reset Command
0xFF	FCSRun	M2M Basic Command	ICM	FCS Execute Command

Table 5.3: M2M Basic Command Payload

Offset	Size	Name	Description
0	4	Timestamp	Microsecond Timestamp

Table 5.4: M2M Comm Data Payload

Offset	Size	Name	Description
0	4	Timestamp	Microsecond Timestamp
4	4	VACS ID	VACS Message ID
8	N	Data[N]	VACS Payload (N Bytes)

Table 5.5: M2M Device Data Payload

Offset	Size	Name	Description
0	4	Driver	Unique Device Driver ID
4	4	Timestamp	Microsecond Timestamp
8	8*N	Data[N]	Array of N Key/Value Pairs

Table 5.6: M2M Device Priority Payload

Offset	Size	Name	Description
0	4	Driver	Unique Device Driver ID
4	4	Priority	Priority of Following Keys
8	4*N	Keys[N]	Array of N Keys

The ICM software generates DeviceReport packets containing sensor measurements and actuator information for use by the FCS. The FCS in turn generates DeviceCommand packets containing actuator and payload instructions for the ICM software. Both of these packet types make use of the "Device Data" payload shown in Table 5.5. The ICM software contains several sensor/actuator device drivers, many of which are capable of handling multiple physical device instances. The Driver field is therefore used to uniquely identify a particular device driver instance. The Timestamp field is used for logging purposes and indicates the time sensor data was acquired or the time a command was generated. The Data field consists of an array of key/value pairs used for transferring the actual sensor/actuator data. Each key is a 32-bit number that identifies the contents of the corresponding value field. Each value is a 32-bit quantity, either a single precision floating point number (F32) or a signed/unsigned integer (I32/U32),

with the particular format being determined by the key. A list of the currently used keys is provided in Table 5.7.

Table 5.7: Device Driver Keys

Key	Description	Units	Format
0x10	Yaw Angle	Degrees	F32
0x11	Pitch Angle	Degrees	F32
0x12	Roll Angle	Degrees	F32
0x20	X-Axis Angular Rate	Degrees / Second	F32
0x21	Y-Axis Angular Rate	Degrees / Second	F32
0x22	Z-Axis Angular Rate	Degrees / Second	F32
0x30	X-Axis Acceleration	g-force	F32
0x31	Y-Axis Acceleration	g-force	F32
0x32	Z-Axis Acceleration	g-force	F32
0x40	Position Latitude	Degrees	F32
0x41	Position Longitude	Degrees	F32
0x42	Altitude Above Mean Sea Level	Meters	F32
0x49	Altitude Above Ground Level	Meters	F32
0x50	Velocity North	Meters / Second	F32
0x51	Velocity East	Meters / Second	F32
0x52	Velocity Down	Meters / Second	F32
0x60	Airspeed	Meters / Second	F32
0x70	GPS Position Fix	Enumeration	U32
0x71	Number of GPS Satellites	Count	U32
0x80	ECU Turbine Speed	Rotations / Minute	F32
0x81	ECU Turbine Exhaust Gas Temp.	Degrees Celcius	F32
0x82	ECU Fuel Pump Voltage	Volts	F32
0x83	ECU Turbine State	Enumeration	U32
0x84	ECU Throttle Position	Percentage	F32
0x90	Barometric Static Pressure	Kilopascals	F32
0x91	Barometric Pitot Pressure	Kilopascals	F32
0xA0	RC Battery	Volts	F32
0xA1	FCS Battery	Volts	F32
0xA2	Motor Battery	Volts	F32
0xB0	Last M2M Transmit Time	Microseconds	U32
0xB1	Last M2M Receive Time	Microseconds	U32
0xF0	Safety Switch Control Mode	Enumeration	U32
0xF1	Safety Switch Fail-Safe Flag	Enumeration	U32
0xFF	Execution Time	Clock Cycles	U32
0x100 + N	Manual PWM for Channel N	Microseconds	U32
0x200 + N	Autonomous PWM for Channel N	Microseconds	U32
0x300 + N	ADC Sample for Channel N	Raw Value	I32
0x400 + N	Analog Voltage for Channel N	Volts	F32

The DevicePriority packets are used to describe the capabilities of each sensor/actuator device driver instance. They provide the FCS with a list of supported properties and commands,

along with a default device priority. Packets of this type will be sent to the FCS whenever the sensor configuration is changed. The FCS can also request sensor priority information from the ICM by sending a DeviceQuery command. The FCS will always send this command at startup to ensure that sensor information remains synchronized.

The particular keys or key/value pairs contained in a given sensor packet are controlled by the source device driver and depend on the measurement capabilities of the corresponding sensor device. The use of this key/value structure allows the sensor interface to become much more flexible and generic. It eliminates the need for the FCS to be aware of the specific sensor devices used, and helps simplify the process of adding new sensor capabilities to the autopilot.

5.5 Sensor Driver Interface

All sensor and actuator device drivers written for the ICM implement a standard API. This API specifies several function prototypes that drivers can implement and defines how the drivers interact with the rest of the system. The use of this API assists in the creation of new sensor drivers and ensures consistent behavior. A list of the device driver functions is provided in Table 5.8. The Reset and Reconfigure functions are required by all drivers while the remaining function handlers are optional.

Table 5.8: Generic Sensor/Actuator Driver Interface

Driver Function	Description
Reset	Removes all device instances. Frees any allocated memory.
Reconfigure	Detects and initializes new device instances. Allocates memory.
PeriodicUpdate	Periodically updates all instances at main FCS rate. Sends new data to FCS.
ContinuousUpdate	Continuously updates all instances during idle time. Sends new data to FCS.
SendPriority	Sends default priority information of all instances to FCS.
Calibrate	Performs manual device calibration for all instances.
HandleCommand	Handles actuator & payload commands from FCS.

Sensor configuration is maintained using a pair of device tables describing the physical sensors and signals connected to the ICM. One table identifies the particular serial device

connected to each ICM UART, while the other table specifies the input to each analog channel and the external voltage gain applied. Multiple sensors of the same or similar type can be connected simultaneously, allowing for the possibility of sensor redundancy. These tables are under user control and can be dynamically adjusted at run-time from the ground station. The user interface for changing the sensor configuration is shown below Figure 5.4.

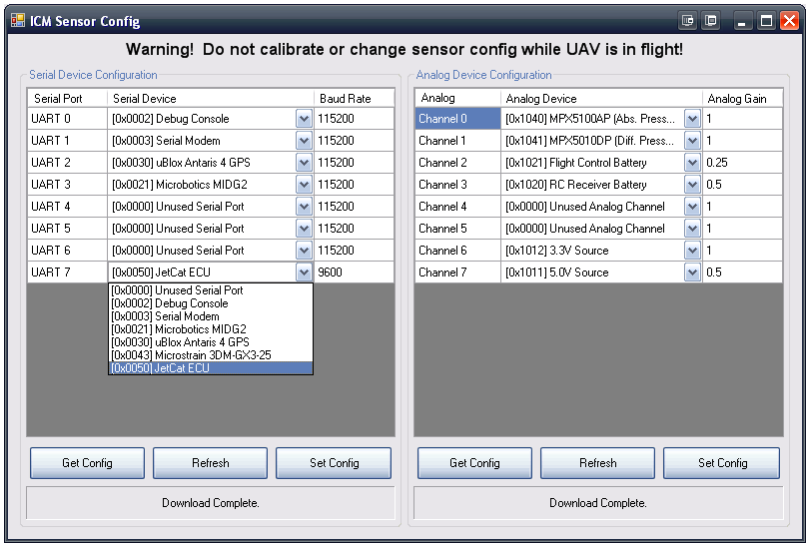

Figure 5.4: GCS Sensor Configuration Window

Just prior to sensor reconfiguration the Reset function will be called for all drivers. This will ensure that any previously allocated memory has been freed, preventing the possibility of memory fragmentation. During reconfiguration, each sensor device driver will query the device tables to determine the number and location of any supported devices. If any supported devices are present, the driver will then initialize its internal variables and allocate memory if necessary. The Calibrate and SendPriority functions will also be called if implemented by the driver.

The Calibrate function is used to initialize the physical sensor devices and calculate any required bias or scale factors. The SendPriority function generates an M2M Device Priority packet for each sensor instance and adds it to the M2M transmit queue. This priority information can then be used by the FCS to determine the preferred sensors for each type of measurement.

For instance, both the absolute pressure sensor and the GPS provide altitude information. However, the altitude measurements from the pressure sensor tend to be more accurate and are therefore given a higher default priority.

The ContinuousUpdate and PeriodicUpdate functions are used to take sensor readings during normal operation. The ContinuousUpdate function is repeatedly called during the asynchronous portion of the ICM main loop. This function is typically used by serial-based sensor drivers to parse the incoming data stream. The PeriodicUpdate function is called during each synchronous iteration of the main loop. It is typically used by analog-based sensor drivers to process and filter new data samples. Both functions will queue M2M Device Report packets if new sensor data is available.

5.6 Sensor Driver Details

The device drivers implemented for some of the more commonly used sensors will be described in more detail below.

5.6.1 uBlox Antaris 4 GPS

The uBlox GPS receiver on the auxiliary board communicates with the ICM via a TTL serial interface operating at 115200 baud. This device provides updated position and velocity information at 4 Hz using the UBX binary protocol. This protocol, shown in Figure 5.5, uses a simple packet format consisting of a six byte header, a variable length payload, and a two byte checksum. All header and payload fields are stored in little-endian byte order. The checksum algorithm used is a modified 16-bit Fletcher checksum with two's complement arithmetic. It is calculated over all bytes between SYNC CHAR 2 and CK_A. During the driver's ContinuousUpdate function, any newly received bytes are assembled using an FSM based parser. Recursive sync recovery is utilized to reduce the negative effects of any corrupt packets.

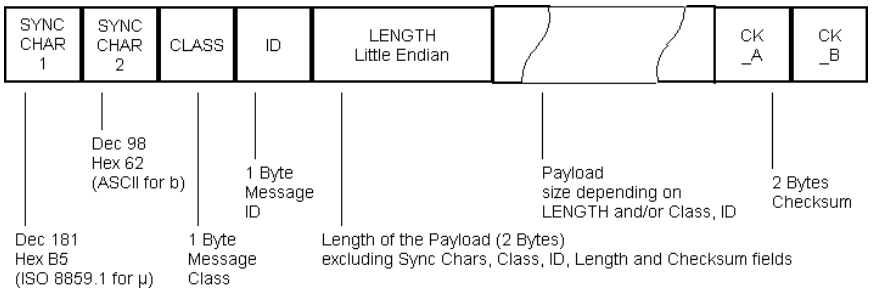

Figure 5.5: UBX Binary Protocol, Antaris GPS [66]

The uBlox device driver currently handles three navigation packets from the GPS: NAV_POSLLH, NAV_VELNED, and NAV_SOL. The NAV_POSLLH packet provides position data in LLA format, the NAV_VELNED packet provides velocity data in NED format, and the NAV_SOL packet provides solution information including the number of satellites used and the type of GPS fix. Whenever a valid packet is received, the relevant fields are extracted and scaled. An M2M Device Report containing the new information is then generated and queued for transmission to the FCS.

5.6.2 Microbotics MIDG II INS/GPS

The Microbotics MIDG II is an external INS/GPS device that interfaces with the ICM using an RS-232 serial interface operating at 115200 baud. Updated position, velocity and attitude information are provided at rates of up to 50 Hz. It implements an advanced Kalman filter for improved measurement accuracy. Data is transferred in the Microbotics Binary Protocol, shown below in Figure 5.6. Packets consist of a four byte header, a payload of up to 255 bytes, and a two byte checksum. All payload fields are stored in big-endian byte order. The checksum algorithm used is a modified 16-bit Fletcher checksum with two's complement arithmetic. It is calculated over all bytes between SYNC 1 and CKSUM 0. An FSM based serial parser is used to assemble the packets during the driver's ContinuousUpdate function call. Like the VACS and UBX parsers, it implements recursive sync recovery.

Figure 5.6: Microbotics Binary Protocol, MIDGII [67]

The MIDG II device driver currently handles the NAV_Sensor, NAV_PV, and GPS_PV messages. Whenever a valid packet is received, the relevant fields are extracted, scaled and converted if necessary. An M2M Device Report containing the new information is then generated and queued for transmission to the FCS. The NAV_Sensor message is used to provide the FCS with 3-axis angular rates, 3-axis acceleration, and orientation information in Euler angle format. The NAV_PV message is used to provide position and velocity information. The GPS_PV message is only used to provide GPS fix and satellite count information.

5.6.3 MPX5010DP (Airspeed)

As mentioned in section 3.5.4, the MPX5010DP sensor is used to measure the impact pressure from the aircraft's Pitot tube. The measured pressure value is presented to the system as an analog voltage that exhibits a linear transfer function. During the driver's PeriodicUpdate function, the newest sampled voltage is converted back to a pressure value. This is done using the reverse transfer function shown below, where V_{Out} is the sensor's output voltage and V_S is the analog supply voltage.

$$P = \frac{V_{Out}}{V_S \cdot 0.09} - \frac{4}{9}$$

During calibration, the driver uses the current sensor voltage to determine the zero pressure bias point and calculate a correction factor. This is used to compensate for the voltage offset at the sensor output. The 5V analog supply is also monitored during operation to correct for temperature induced supply drift and improve the accuracy of the conversion.

Once an impact pressure reading is obtained, it is used to determine the aircraft's present airspeed. This is accomplished using the calibrated airspeed (CAS) formula for subsonic speeds [68].

$$CAS = a_0 \sqrt{5\left[\left(\frac{P}{P_0}+1\right)^{\frac{2}{7}}-1\right]}$$

P = Impact Pressure (kPa)

P_0 = Standard Pressure at Sea Level (101.325 kPa)

a_0 = Standard Speed of Sound at 15°C (661.5 knots)

This equation is valid over the entire operating range of the pressure sensor, allowing airspeeds of up to 243 knots to be calculated. Figure 5.7 shows the relationship between impact pressure and calibrated airspeed up to the limit of the sensor. Initially the slope is very steep, with large changes in airspeed corresponding to small changes in pressure. At higher airspeeds, the pressure increases more rapidly with increasing airspeed.

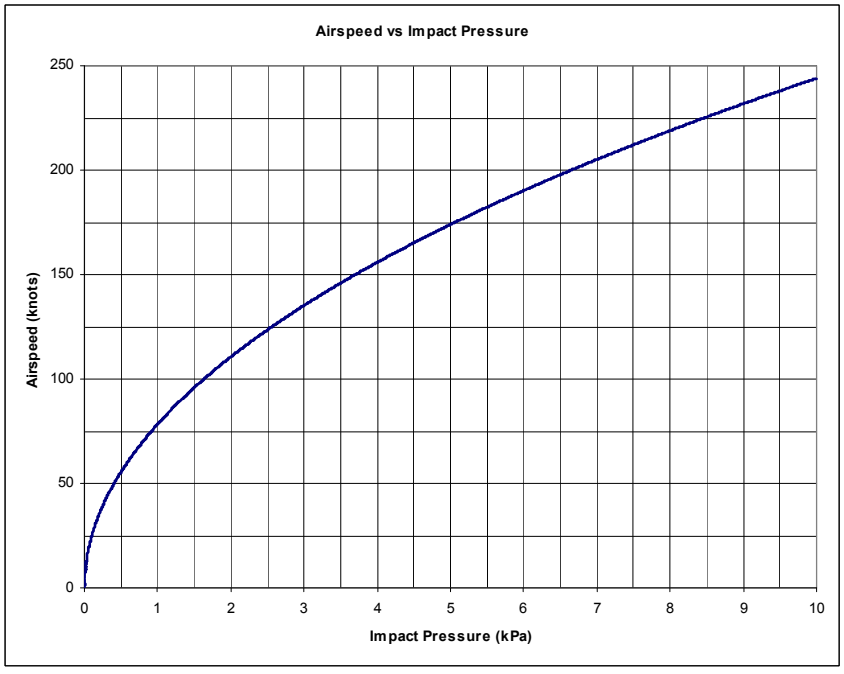

Figure 5.7: Calibrated Airspeed vs. Impact Pressure

Due to the non-linear relationship between pressure and airspeed, the accuracy of the calculation varies with airspeed. The effective airspeed resolution for a 16-bit reading is shown below in Figure 5.8. As expected, resolution is poor at low airspeeds but rapidly improves as airspeed increases. At 5 knots the effective resolution is already 0.1 knots per LSB.

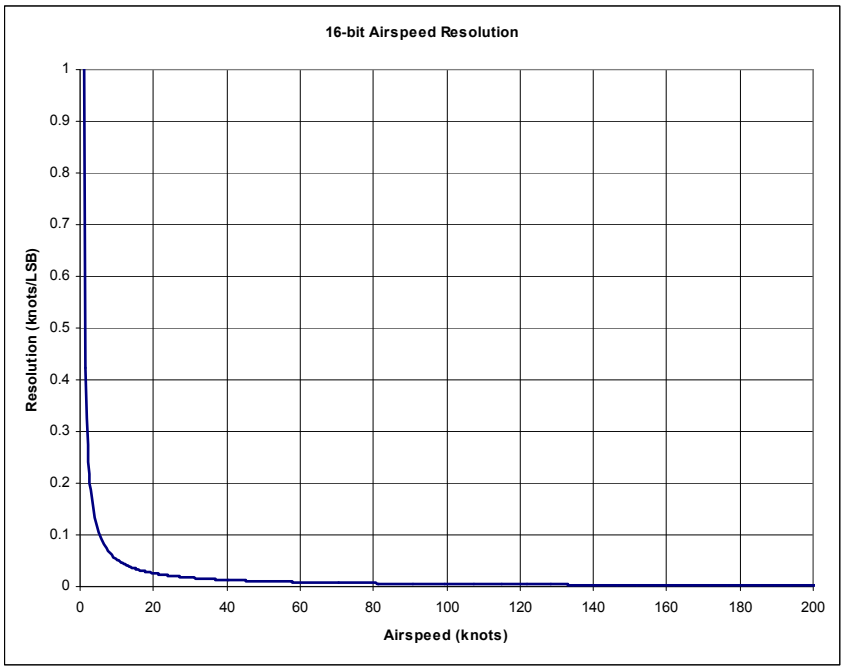

Figure 5.8: Effective Airspeed Resolution vs. Airspeed

This driver also provides an optional 1st order IIR low pass filter that can be enabled to reduce any vibration induced noise not removed by hardware filtering. This filter will result in a maximum group delay of 75 ms and can be disabled if reduced latency is required. Also, since the effect of noise will decrease at higher airspeeds, filtering is only likely to be useful at speeds below 10-20 knots.

5.6.4 MPX5100AP (Altitude & Climb-Rate)

The MPX5100AP absolute pressure sensor is used to measure static barometric pressure. The measured static pressure is presented to the system as an analog voltage, as described

previously in section 3.5.4. Reversing the MPX5100AP transfer function yields pressure in terms of the sensor output voltage, V_{Out}, and the analog supply voltage, V_S.

$$P = \frac{V_{Out}}{0.009 \cdot V_s} + \frac{0.095}{0.009}$$

The barometric formula, also referred to as the exponential atmosphere formula, can be used to compute the pressure, P, as a function of height and atmospheric layer [69].

$$P = P_b \left[\frac{T_b}{T_b + L_b \left(h - h_b \right)} \right]^{\frac{g_0 \cdot M}{R^* \cdot L_b}}$$

h = Height above sea level (meters)

h_b = Height at bottom of layer b (meters)

P_b = Static pressure at bottom of layer b (kPa)

T_b = Standard temperature of layer b (K)

L_b = Standard temperature lapse rate of layer b (K/m)

R^* = Universal gas constant for air: 8.31432 N·m/(mol·K)

g_0 = Gravitational acceleration (9.80665 m/s^2)

M = Molar mass of Earth's air (0.0289644 kg/mol)

Solving the barometric formula for h yields the following equation:

$$h = h_b + \left[\left(\frac{P}{P_b} \right)^{\frac{-R^* \cdot L_b}{g_0 \cdot M}} - 1 \right] \frac{T_b}{L_b}$$

Within the first atmospheric layer (0 to 11000 meters), P_b = 101.325 kPa, T_b = 288.15 K, L_b = -0.0065 K/m, and h_b = 0 m. This reduces the altitude equation to:

$$h = -44330.77 \left[\left(\frac{P}{101.325} \right)^{0.190263} - 1 \right]$$

Because this autopilot platform will not be flown at altitudes above the first atmospheric layer, the reduced equation shown above can be used to determine altitude based on static

pressure. During the MPX5100AP driver's PeriodicUpdate function, the newest analog voltage sample will be used to calculate pressure. The calculated pressure will then be used to determine the aircraft's altitude above mean sea level (AMSL). The relationship between pressure and altitude is shown in Figure 5.9, while the effective altitude resolution is shown in Figure 5.10.

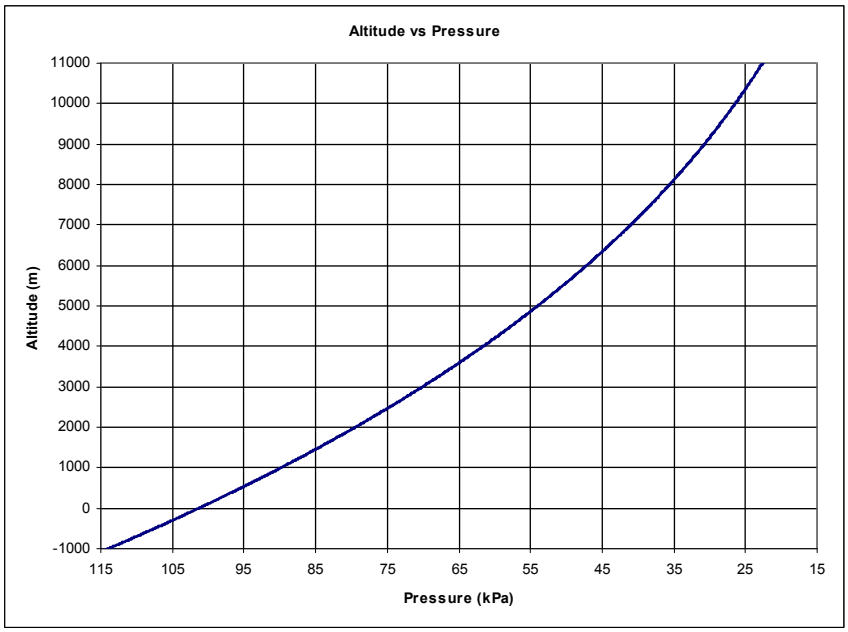

Figure 5.9: Altitude AMSL vs. Barometric Pressure

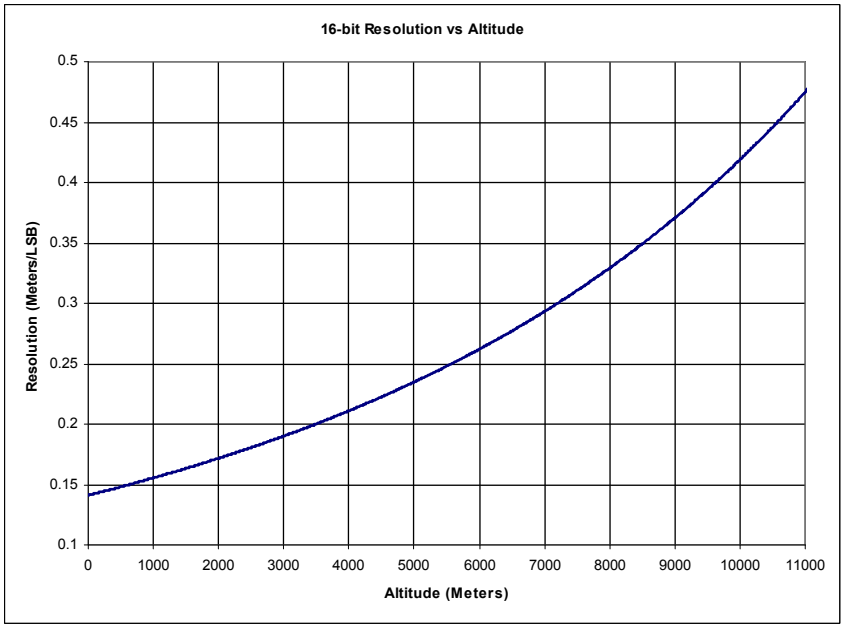

Figure 5.10: Effective Altitude Resolution vs. Altitude

As shown, this barometric altitude conversion provides a fairly high degree of accuracy. However, the FCS also requires climb-rate information in order to perform altitude hold. Due to the high sample rate, calculating climb-rate directly from changes in measured altitude would yield unusable results. At ground level, the altitude resolution of 0.14 meters would result in a calculated climb-rate precision of 28 m/s. To calculate more accurate climb-rate values an α-β filter is used.

An α-β filter is a simplified form of linear state observer that assumes constant velocity linear motion. It is functionally similar to a steady-state Kalman filter with unchanging process and measurement noise covariance. Pre-computed gain constants are used, as apposed to the Kalman filter's dynamically updated gains [70, 71].

An α-β filter consists of two stages, prediction and correction. The prediction stage calculates an estimate of position and velocity based on the previous filter outputs. The correction stage updates the estimates using a newly measured position value. In this case, position refers to altitude and velocity refers to climb-rate. The equations used to implement the filter are shown below.

$$
\begin{aligned}
x_p(k) &= x_s(k-1) + T v_s(k-1) \\
v_p(k) &= v_s(k-1) \\
x_s(k) &= x_p(k) + \alpha\left(x_0(k) - x_p(k)\right) \\
v_s(k) &= v_p(k) + \frac{\beta}{T}\left(x_0(k) - x_p(k)\right)
\end{aligned}
$$

α = Alpha Gain Constant β = Beta Gain Constant T = Time Step (5 ms)

x_p = Predicted Position x_s = Smoothed Position x_0 = Measured Position

v_p = Predicted Velocity v_s = Smoothed Velocity

Mission logs from previous test flights were used to determine the optimal filter gains. First, an ideal climb-rate estimate was computed from the barometric altitude measurements

using a zero-lag FIR filter in MATLAB. Several time-delayed versions of this estimate were then created. An iterative error reduction routine in MATLAB was used to determine the α-β gains that would yield the best climb-rate match for each of the delayed versions. It was found that a velocity delay of at least 200 ms was necessary to calculate accurate climb-rate values. Although this delay is significant, it is still an improvement compared to the corresponding delay in the previous generation autopilot. The final gains chosen were: $\alpha = 0.041015625$ and $\beta = 0.000732421875$.

In addition to calculating climb-rate, this filter also calculates a smoothed altitude estimate. This smoothed altitude output does not exhibit the large time delay present in the climb-rate output. In most cases, improved altitude accuracy is provided with negligible delay. In cases of rapid vertical acceleration however, there can be a temporary lag effect of up to 50 ms. This is due to the constant velocity assumption of the filter model. In practice, this lag has only been observed to cause a maximum error of 2 feet. However, this error may increase when faster, more maneuverable aircraft are flown. In this case the altitude smoothing can be bypassed, returning the unfiltered altitude values.

5.6.5 Battery Voltage Monitor

The FCS battery and RC battery voltages are monitored using resistor voltage dividers connected to two of the system's analog inputs. During this driver's PeriodicUpdate function the sampled voltages are scaled and filtered. Monitoring these voltages allows the FCS to determine when a battery is low using a simple voltage threshold check. When this condition occurs, the FCS can automatically enter a "return-home" mode and fly the plane back to a set rally point. A simple 1^{st}-order lag filter with a time constant of 1 second is used to smooth out any measured

voltage fluctuations. Without this filtering there is the possibility that the FCS would enter "return-home" mode unnecessarily due to brief load transients affecting the battery voltage.

5.7 Issues Encountered

As mentioned previously, the ICM software was developed using the Xilinx EDK and supporting utilities. The use of these tools and libraries initially appeared highly beneficial; they should theoretically reduce development time and allow the user to focus more on writing the application code. Unfortunately, several problems were encountered during development that resulted in poor performance, incorrect behavior, and even intermittent failure. These issues were originally assumed to be due to user error, and much time was spent debugging and troubleshooting the software. However, it was eventually discovered that many of the issues were actually due to bugs in the Xilinx provided tools and libraries. In the end, several workarounds were necessary and many of the Xilinx provided libraries had to be rewritten to correct serious flaws and performance deficiencies. Some of the issues and subsequent workarounds will be described below.

In order for optional MicroBlaze features, such as hardware multiply and floating point instructions, to be used by the compiler, the appropriate GCC flags must be set. This is normally handled transparently by the Xilinx build process. However, a bug was introduced in version 10.1i of the Xilinx EDK that prevents the compiler flags from being set correctly for Spartan-3 based projects. This means that even if the affected hardware instructions are enabled, the compiled software will not make use of them. Instead, several tens of kilobytes of software emulation functions are pulled into the compiled binary. This can drastically reduce performance if the affected operations are used frequently. Manually adding the necessary compiler flags to the software project allows this bug to be circumvented [72].

Another bug is present in the MicroBlaze version of Newlib and affects some of the C standard library functions. One such function is rand(), the pseudo-random number generator function. When this function is called, an internal null-pointer is dereferenced and written to, resulting in corruption of the MicroBlaze interrupt vector. If interrupts are enabled, the next interrupt to occur will result in a hard lock [73]. This bug was very difficult to track down due to the seemingly random occurrence of the lockups. Originally, the rand() function was used during development to generate data for testing the M2M bus. Once the nature of the bug was discovered, the use of rand() was replaced with manually generated test data.

The Xilinx provided UARTLite drivers also contain a bug that can occur when transmitting data in interrupt mode. To ensure atomicity when accessing internal buffers, the low-level drivers will mask the UARTLite interrupts when exchanging data with the user program. The bug itself is due to the way that the interrupts are masked and will manifest itself whenever a "Transmit Complete" interrupt occurs while being masked. After the interrupts are unmasked, the transmit handler will not be invoked and any pending transmissions will stall indefinitely. The only reliable way found to correct this issue was to rewrite the UARTLite drivers. Interrupt masking was changed to use the hardware interrupt controller's enable/disable registers, rather than the UARTLite's internal control register mask bits. This ensured that the interrupt handler was always called when necessary.

Although technically not a bug, the low-level interrupt handlers for the Xilinx Interrupt Controller and UARTLite were not well optimized. This resulted in excessive CPU usage when continuously receiving data from one or more UARTs. In particular, handling a single received character would incur a total CPU overhead of approximately 500 clock cycles. At 50 MHz, one UARTLite configured for 115200 baud could therefore potentially consume over 11% of the

CPU time in interrupt handling alone. Completely rewriting the relevant low-level drivers reduced the combined interrupt overhead to approximately 150 clock cycles per received character, an improvement of ~70%. Further gains are potentially possible by modifying the UARTLite IP core itself to generate fewer interrupts via increased hardware buffering or by transferring data using DMA.

One final, albeit minor, issue involves the standard output printing function printf(). During development it is often beneficial to periodically print the contents of internal variables to the console, to aid in debugging. This is typically accomplished using the printf function. However, there are a few aspects of the Newlib version of printf that limit its usefulness in this application. The main problem is due to the blocking nature of the Xilinx stdio implementation. Printing a string to the serial console uses polled I/O, resulting in non-trivial delays once the 16-byte UARTLite FIFO is filled. This obviously has a negative affect on system timing. The second, less critical, problem is simply due to printf's size; it requires approximately 50kB of code memory and several kilobytes of stack and heap space. These memory requirements invariably result in poor cache performance, with large portions of the main program being evicted during the printf call. These two problems were rectified by writing a light-weight version of printf that makes use of buffered, interrupt driven I/O. The custom version only requires 2kB of code memory and typically executes over 10 times faster than the standard version. It only lacks support for floating point numbers.

Chapter 6: System Performance Analysis

Extensive in lab testing was performed on all aspects of the new autopilot platform to verify safety and correct operation prior to flight. This included individual tests of each hardware sub-system, including the safety switch, wireless radio link, sensors, analog signal conditioning and voltage regulation components. The software was also thoroughly tested and debugged to minimize the risk of failure. In addition to unit testing, full system tests were conducted in lab using hardware-in-the-loop simulation. Over 70 such simulations were performed to verify proper functionality of the complete autopilot system.

Several fully autonomous test flights were conducted once sufficient lab testing had been performed and confidence was gained. These flights were used to verify correct operation and determine the performance of the system during real world conditions. Highly detailed logs were recorded containing sensor readings, actuator control values, internal FCS parameters and CPU usage information. The primary purpose of gathering this data was to determine sensor measurement accuracy and to perform system timing analysis. During the later flights, PID parameter tuning was also conducted to improve flight performance. The results of some of these test flights will be presented in the following sections.

6.1 Sensor Comparisons

Several of the flights focused on performing comparisons of various sensor devices. One such flight tested the relative accuracy of the MIDG II INS, uBlox GPS, and MPX5100AP pressure sensor with regards to altitude measurements. During this flight the autopilot attempted

to maintain a constant altitude of 305 meters while following a rectangular flight path. However, the aircraft tended to slightly gain altitude during turns due to excessive elevator action. These altitude variations were used to help determine the accuracy of the three sensors. The relevant altitude data is shown below in Figure 6.1.

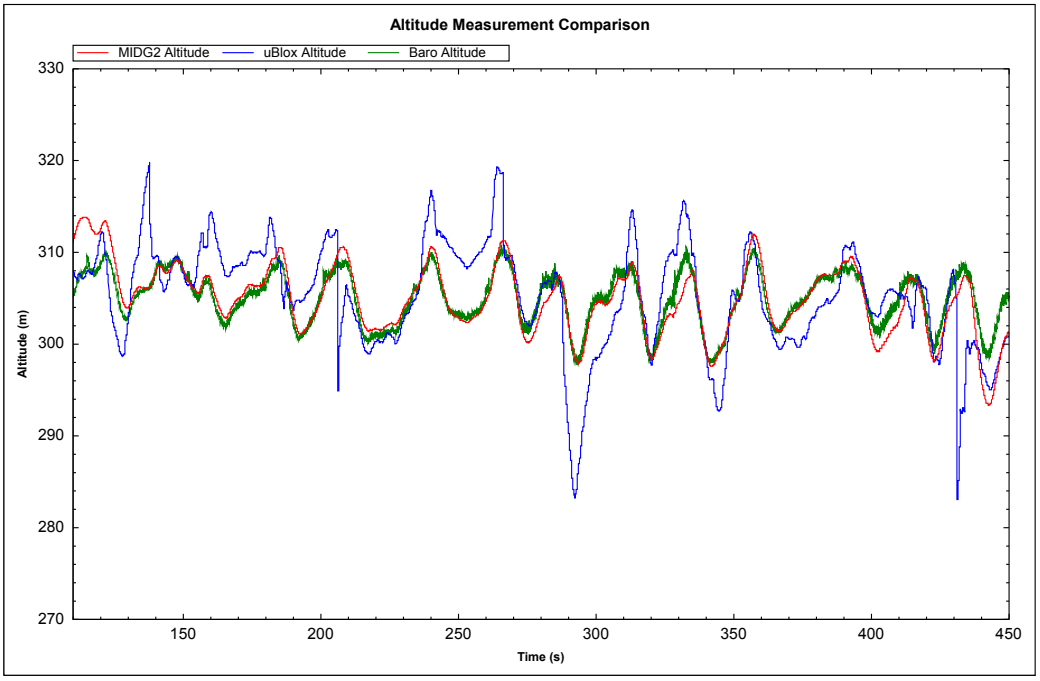

Figure 6.1: Altitude Measurement Comparison - MIDG II, uBlox GPS, and MPX5100AP

The altitude readings of the MIDG II and MPX5100AP tracked each other fairly well during the majority of the flight. However, the uBlox GPS tended to exhibit relatively exaggerated changes in altitude values, with errors of up to 25 meters compared to the other two sensors. The MPX5100AP measurement showed reduced lag compared to the MIDG II, but exhibited slightly higher quantization noise. This noise was present because the MPX5100AP α-β filter was not yet implemented at the time these measurements were taken. Nonetheless, the noise remained below ±0.31 meters at all times.

Another test flight was performed once the α-β filter was implemented to verify correct operation of the filter. The target altitude was varied during this flight in order to test the

autopilot's glide-slope and altitude-hold performance. Autopilot PID parameter tuning was also conducted during this flight. Figure 6.2 shows the altitude readings from the MIDG II and MPX5100AP during the first 10 minutes of flight.

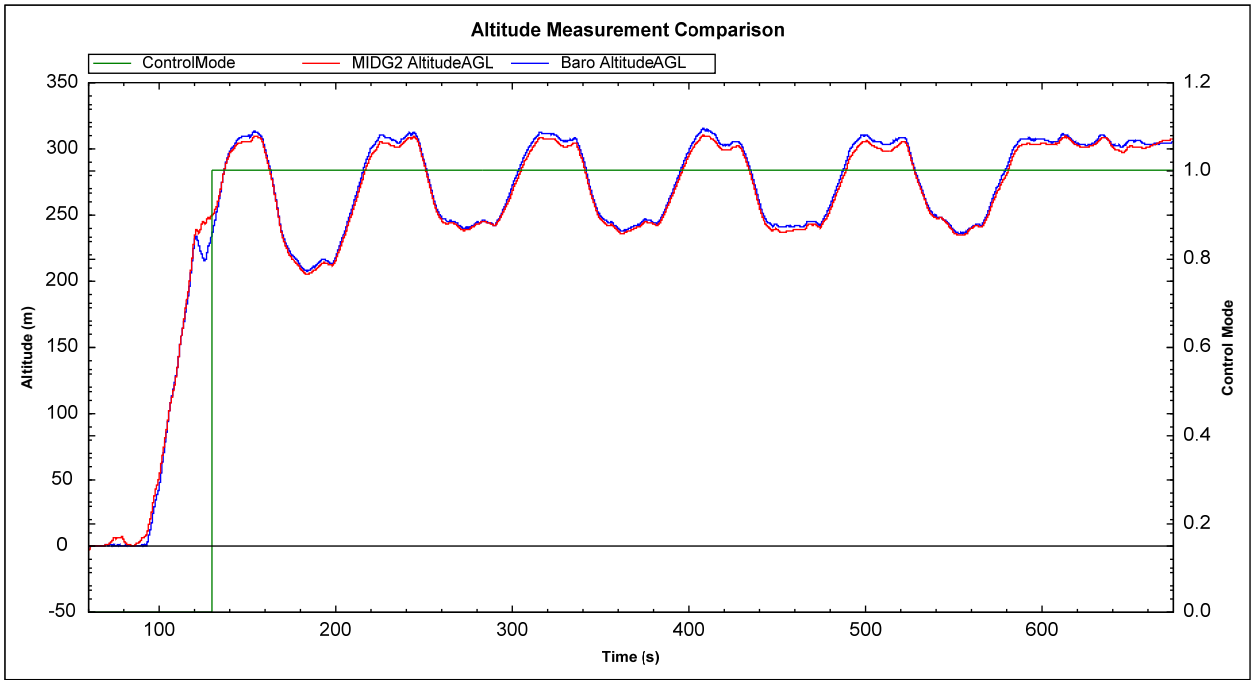

Figure 6.2: Altitude Measurement Comparison - MIDG II vs. MPX5100AP with α-β Filter

Like the previous flight, the altitude readings of the MIDG II and MPX5100AP were closely matched, with the MIDG II exhibiting a small delay. However, one rather large discrepancy in the two measurements can be observed at around 120 seconds in the log. This was after takeoff and just prior to switching from manual operation to autonomous flight. The MPX5100AP altitude reading indicated a 3.5 m/s descent lasting 5 seconds, while the MIDG II indicated that the plane continued to climb. Further analysis of the log showed that the plane exhibited a sharp decrease in pitch just prior to this discrepancy. It is therefore highly likely that the MPX5100AP altitude measurement was more accurate. Figure 6.3 shows a better view of the altitude discrepancy along with aircraft pitch. Pitch angle is shown to rapidly decrease from +50 degrees to -20 degrees just before the sensor disagreement.

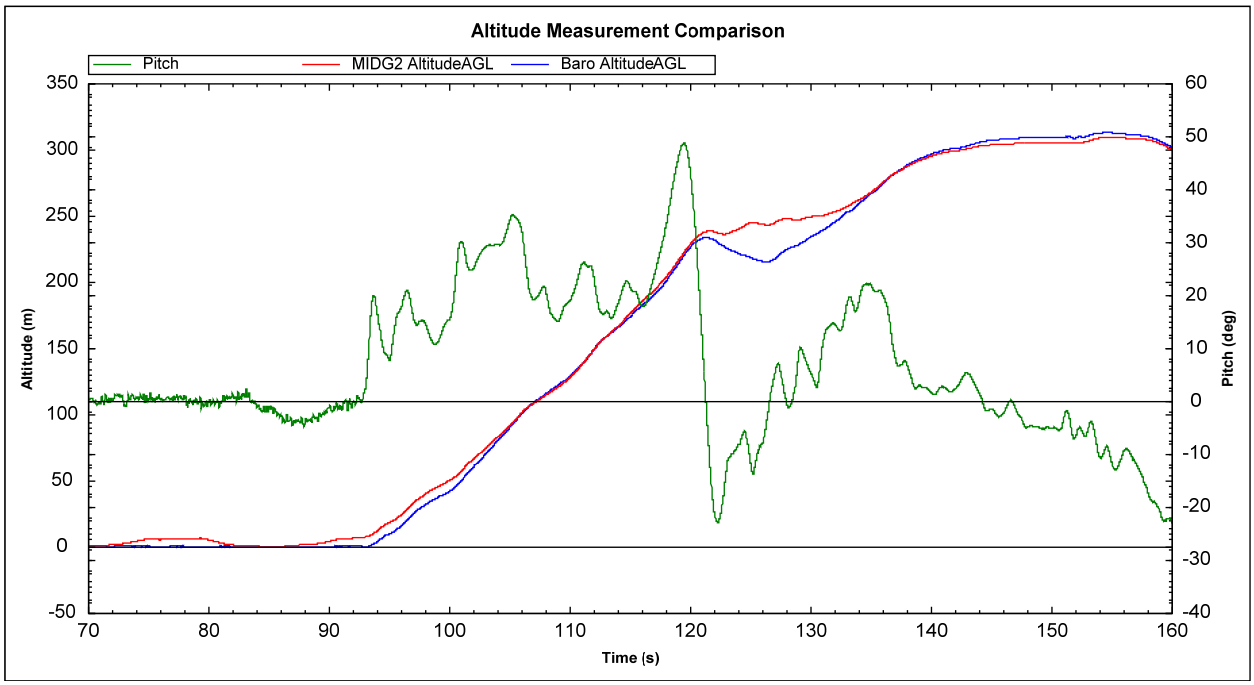

Figure 6.3: Altitude Measurement Discrepancy - MIDG II vs. MPX5100AP with α-β Filter

6.2 Flight Performance

Autopilot parameter tuning and performance analysis were conducted during the last few test flights. This included testing of altitude-hold, airspeed-hold, and flight path tracking. Figure 6.4, shows the aircraft's altitude transition performance when glide-slope mode was enabled. During this mode, the FCS handles altitude changes by smoothly adjusting the target climb-rate of the aircraft. As shown, target altitude was varied between 800 feet and 1000. Some overshoot and undershoot occurred when transitioning to the level segments of flight, but was limited to approximately ±25 feet. Further PID tuning could likely improve the glide-slope performance.

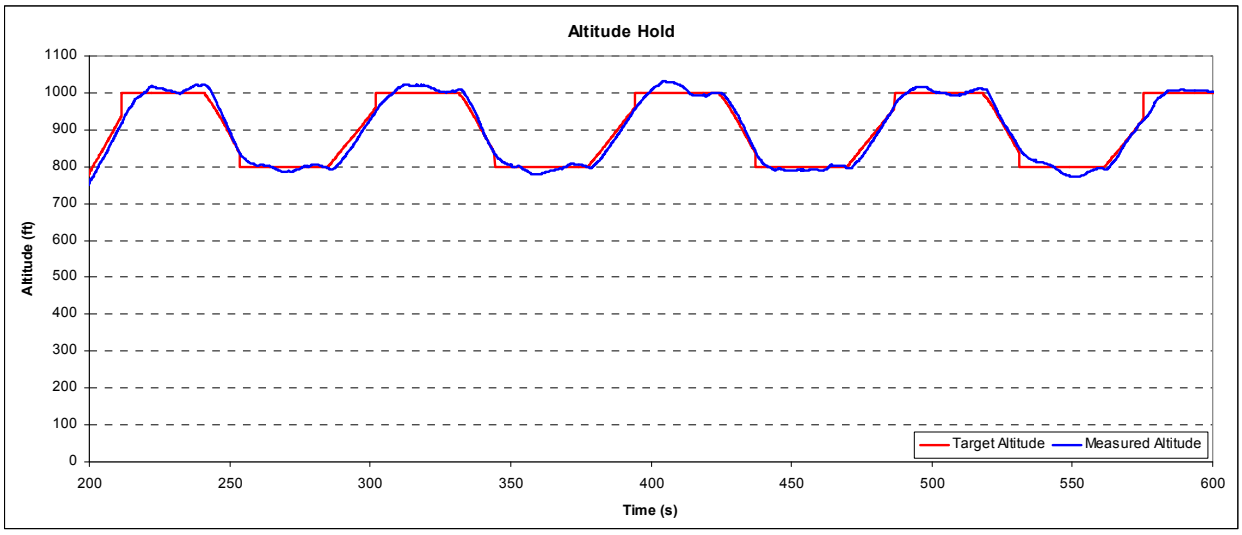

Figure 6.4: Altitude Hold Performance – Glide-Slope Tracking

The performance of altitude hold and transition without glide-slope mode is shown in Figure 6.5 below. Altitude oscillations and undershoot are lower compared to when glide-slope was enabled. These variations are partially due to changes in target airspeed during the level segments of flight.

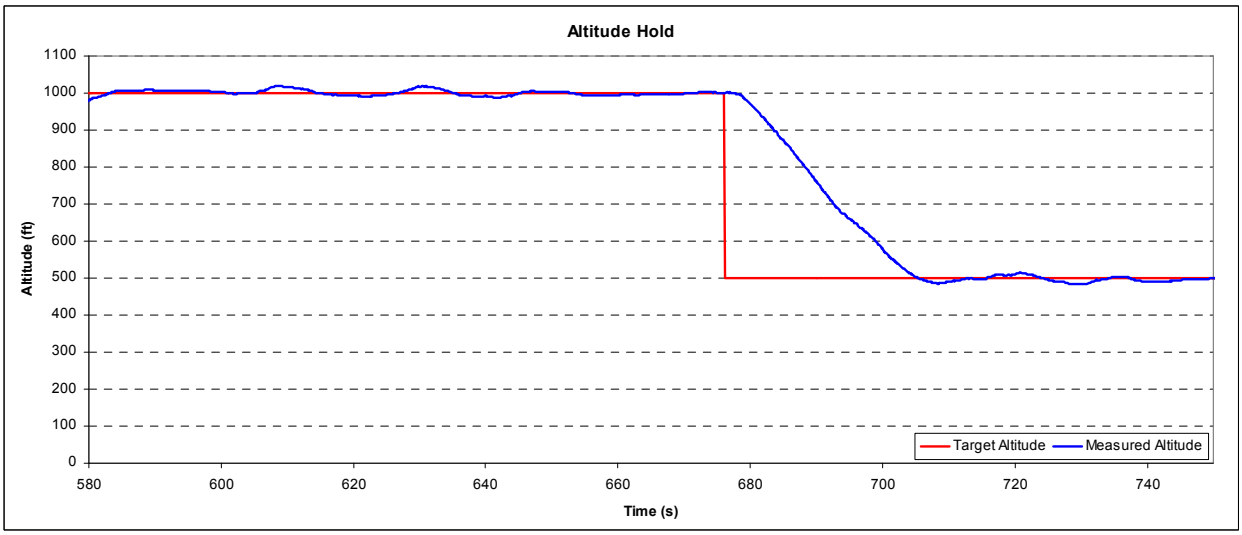

Figure 6.5: Altitude Hold Performance – Large Transition

Airspeed hold and transition performance was also tested during this flight. Target airspeed was varied between 35 knots and 50 knots while altitude was held constant. Figure 6.6 shows the UAV's airspeed response during this phase of the test. Airspeed oscillations of up to 5

knots are apparent, but no undershoot occurred during transition. Subsequent analysis of the log file showed that the oscillations were largely due to an overly high proportional gain in the throttle PID control loop. Decreasing the proportional term and slightly increasing derivative would likely improve performance.

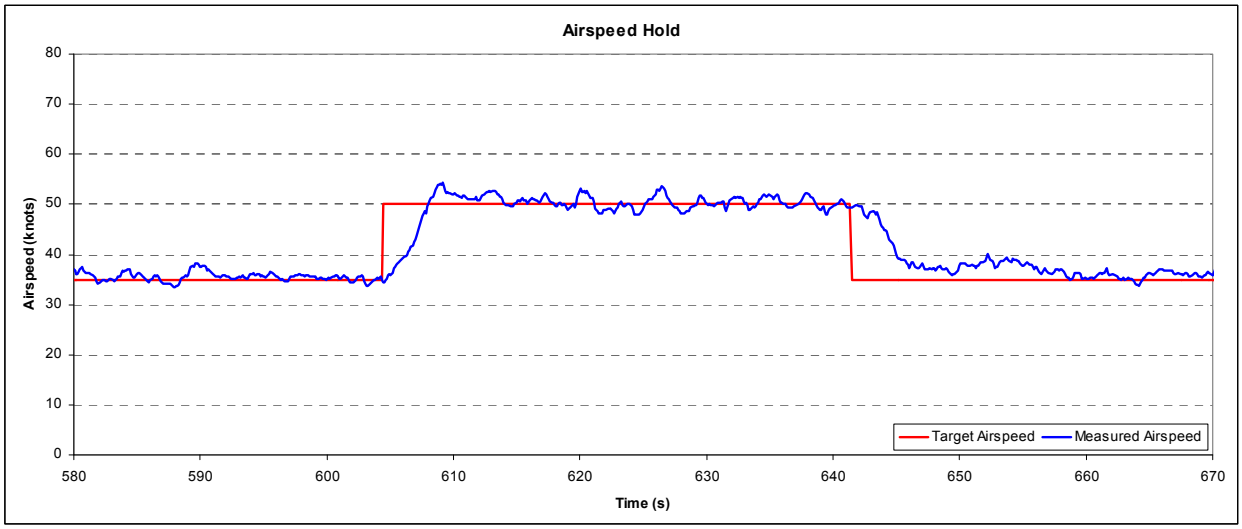

Figure 6.6: Airspeed Hold and Transition Performance

A rectangular, counter-clockwise flight path was used during these tests and the autopilot's cross-track navigation mode was enabled. The flight path and tracking performance are shown below in Figure 6.7. The plane tended to follow the path closely, only overshooting slightly during the turn in the north-west corner.

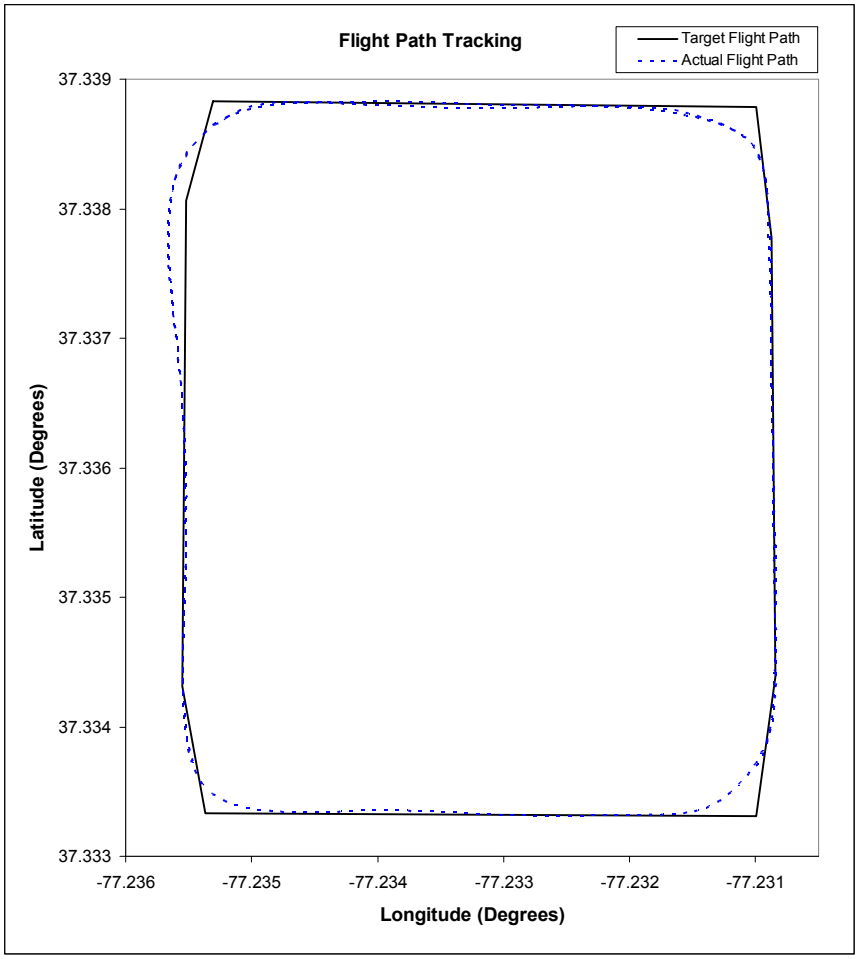

Figure 6.7: Flight Path Tracking Performance

6.3 CPU Utilization

Extensive CPU timing information was collected during the last test flight discussed above. This required some modification to the ICM software, but allowed for analysis of CPU utilization requirements. During normal operation, the ICM software will continuously run sensor parsing and command handling tasks during any idle time. This minimizes the average ICM ⇔ FCS response time, but technically results in 100% CPU usage. The absolute minimum CPU time requirements could be determined by moving all idle tasks into the periodic loop. This would restrict all events to a 200 Hz update rate, but would also negatively impact response time. Instead, the 200 Hz loop was split into 20 slices of 250 µs each. Idle tasks were then limited to one invocation per slice, yielding a maximum idle update rate of 4 kHz. Periodic tasks

were still executed at 200 Hz. This yields a slightly more pessimistic view of the required CPU time, but minimizes the impact on response time.

Figure 6.8 shows the total number of active clock cycles during each complete iteration of the 200 Hz loop. Only the first 160 seconds are shown below, but the distribution remained nearly constant throughout the entire test flight. Loop execution typically required between 23600 and 37400 cycles, with infrequent spikes of up to 46200 cycles. At a CPU speed of 50 MHz, these figures are equal to utilization of 9.44%, 14.96%, and 18.48% respectively.

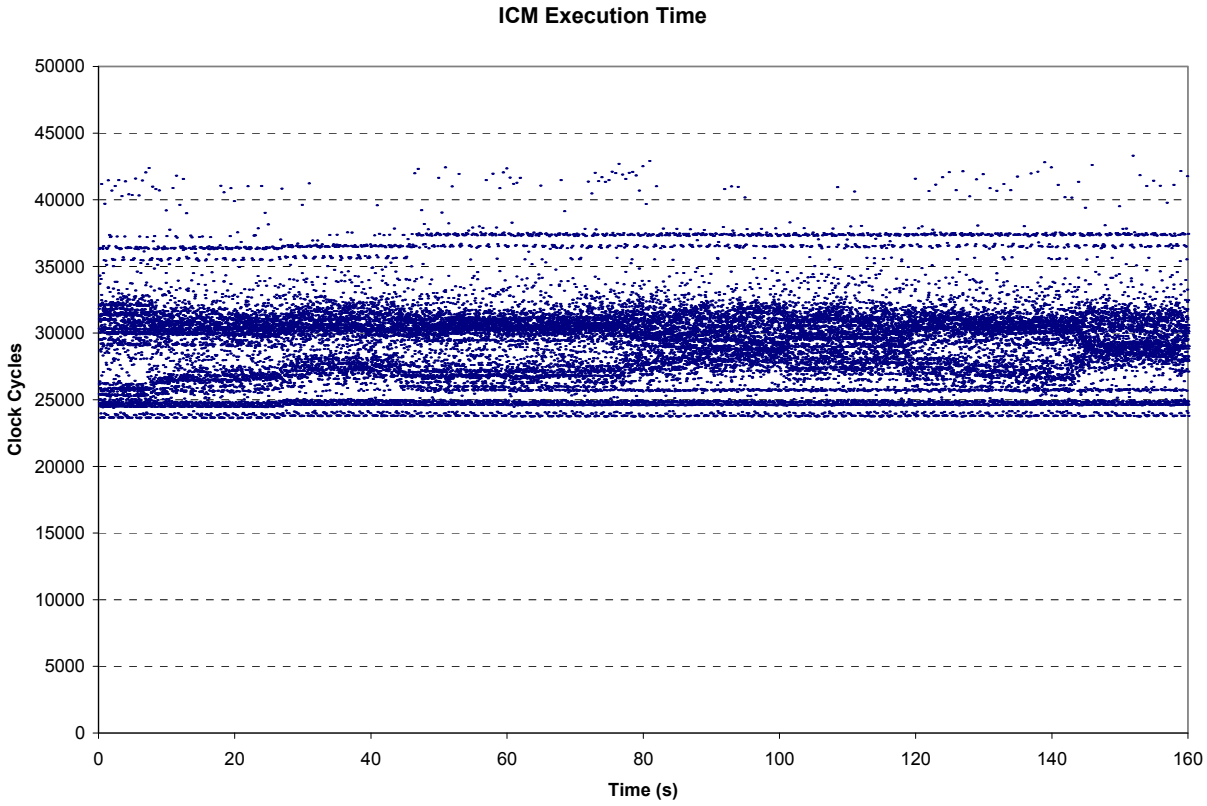

Figure 6.8: Active Execution Time of ICM Loop

During this flight test all M2M data between the two modules was logged and timestamped. This allowed the round-trip time of the entire autopilot control loop to be measured. This was accomplished using the timestamps in M2M Device Report and Device Command packets. At the beginning of the ICM's periodic loop a device report is created

containing the newest ADC samples. When the FCS control loop is finished executing it creates a device command with the actuator output values. Calculating the difference between the arrival time of the actuator command and the previous ADC report timestamp yields the control loop latency. This information is shown below in Figure 6.9. The relatively large increase at ~46 seconds corresponds to the switch between manual and autonomous operation and reflects the time needed to run the FCS control and navigation routines. As shown, overall latency is typically between 1.2 ms and 1.6 ms during autonomous operation.

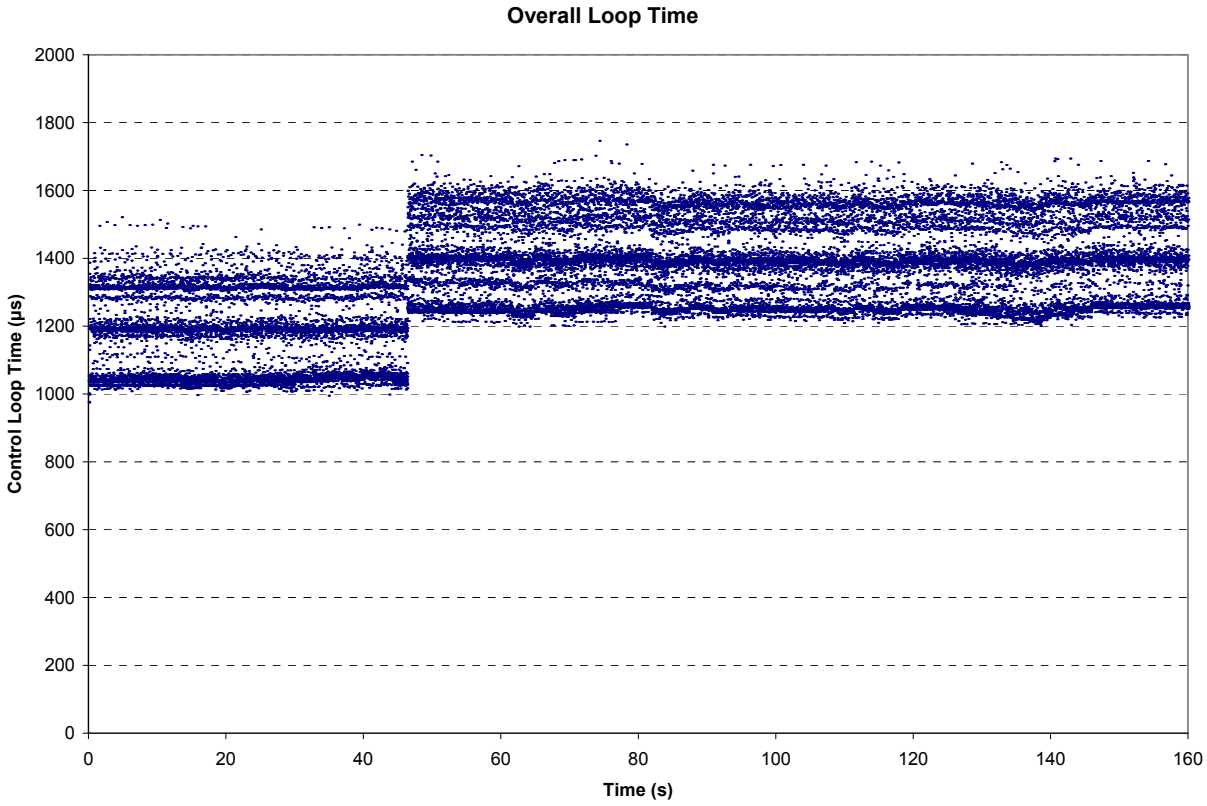

Figure 6.9: ICM → FCS → ICM Round Trip Latency

The FCS software also records the actual time taken to execute its various sub-routines. The execution time of the control and navigation routines is shown below in Figure 6.10. Once autonomous control is engaged, these routines require between 230 µs and 340 µs to complete. This is between 5% and 6.8% of the total 5 ms available during each iteration of the 200 Hz

loop. Note that this only includes the time needed to process the control algorithms; it does not include time used for logging, data transfer or operating system overhead.

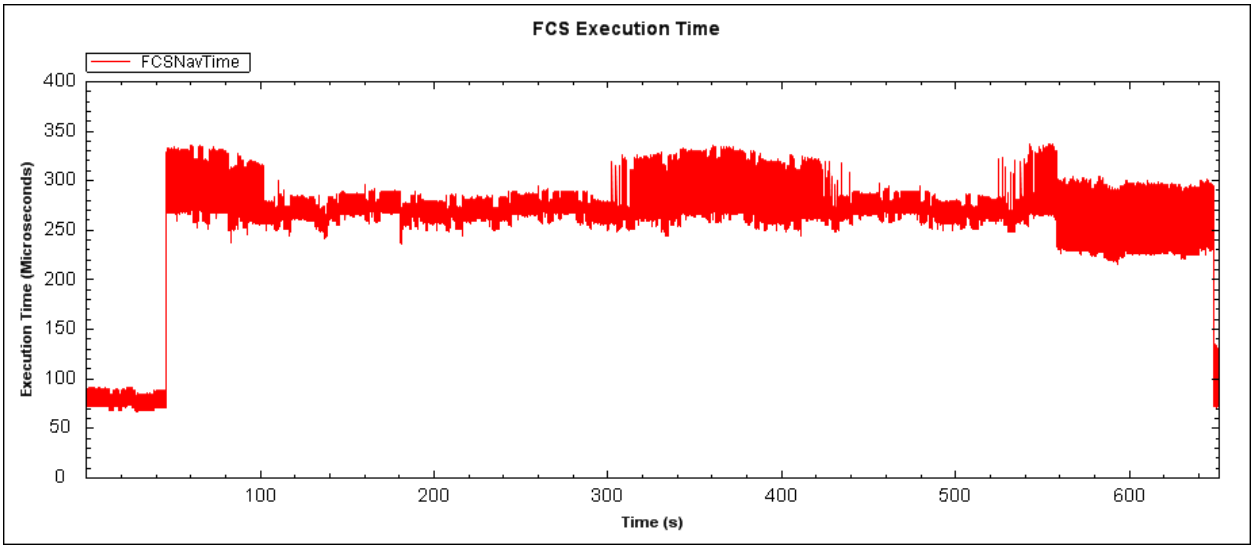

Figure 6.10: Execution Time of FCS Control & Navigation Routines

Chapter 7: Conclusion & Future Work

7.1 Conclusion

The autopilot platform developed for this thesis successfully met or exceeded all design goals and requirements while addressing limitations of the previous generation autopilot. Increased flexibility is afforded due to the dual-FPGA architecture and multi-board hardware design. Processing performance was substantially improved, with ample resources for future use. The FCS control algorithms can now execute at 200 Hz while using less than 7% of the available CPU time. Worst case control loop latency is well under 2 milliseconds. The ICM software also used less than 20% of its CPU time. If the processing power of the current configuration is ever exhausted, the Spartan-3 Mini-Module can always be replaced with a second FX12 Mini-Module without significantly increasing the total system cost.

Other aspects of the system were also upgraded. Wireless communications performance was improved, allowing for telemetry update rates of over 30 Hz. Non-volatile storage was added, allowing for greater mission logging capabilities. The resolution and accuracy of all analog measurements increased significantly and compatibility with previously used sensor devices was maintained. Safety was also improved via the inclusion of an onboard actuator safety-switch and the addition of power supply protection components. System dimensions did not increase significantly and airframe compatibility was also preserved.

7.2 Future Work

There is still room for significant improvements to be made to the autopilot. The most obvious would be the development and testing of more sophisticated control algorithms that can make use of the system's improved computational capabilities. Multi-UAV coordination and collaboration could also be investigated as a possible use for these processing resources. The mesh-networking capabilities of the new radio modems should also be explored as a possible replacement for the current VACS wireless protocol.

Another possibility would be the development of a custom onboard IMU/INS solution. This would remove the need for an external IMU, achieving significant cost and space savings. An updated auxiliary board could be designed that incorporates a 3-axis accelerometer, 3-axis gyro, and 3-axis magnetometer. A more recent GPS receiver with improved altitude accuracy and faster time to first fix could also be added. Sensor fusion algorithms, such as extended Kalman filtering, could then be implemented on the ICM.

References

[1] R. H. Klenke, J. McBride, and H. Nguyen, "A Reconfigurable, Linux-based, Flight Control System for Small UAVs," in *Proceedings of AIAA Infotech@Aerospace 2007 Conference and Exhibit*, Rohnert Park, California, May 2007.

[2] Q. Cheng, "The Development of an FPGA-Based Autopilot for Unmanned Aerial Vehicles," Master's thesis, Virginia Commonwealth University, August 2006.

[3] Atmark Techno. Suzaku-S Specifications. [Online]. Available: http://www.atmark-techno.com/en/products/suzaku/suzaku-s/specs

[4] R. C. DeMott II, L. B. Mize IV, and R. H. Klenke, "Control Algorithms Used in the VCU Rotor-Wing Flight Control System," in *Proceedings of AIAA Infotech@Aerospace*, Atlanta, Georgia, April 2010.

[5] W. C. Sleeman IV, "The Development of a Linux and FPGA Based Autopilot System for Unmanned Aerial Vehicles," Master's thesis, Virginia Commonwealth University, May 2007.

[6] R. H. Klenke, W. C. Sleeman IV, and M. A. Motter, "A High-Throughput Processor for Flight Control Research Using Small UAVs," in *25th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*, June 2006.

[7] MicroPilot - Products - MP2028 Series Autopilots. [Online]. Available: http://www.micropilot.com/products-mp2028-autopilots.htm

[8] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. A. Goodrich, "Autonomous Vehicle Technologies for Small Fixed-Wing UAVs," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, pp. 92–108, January 2005.

[9] Procerus Technologies - Kestrel Autopilot. [Online]. Available: http://www.procerusuav.com/productsKestrelAutopilot.php

[10] Cloud Cap Technology - Piccolo Family of Small Integrated Autopilots. [Online]. Available: http://www.cloudcaptech.com/piccolo_system.shtm

[11] Guided Systems Technologies. [Online]. Available: http://www.guidedsys.com/piccolo-autopilot.php

[12] H. Christophersen, R. W. Pickell, A. A. Koller, S. K. Kannan, and E. N. Johnson, "Small Adaptive Flight Control Systems for UAVs Using FPGA/DSP Technology," in *AIAA 3rd Unmanned Unlimited Technical Conference*, September 2004.

[13] Adaptive Flight - FCS20. Adaptive Flight. [Online]. Available: http://www.adaptiveflight.com/products_fcs20.html

[14] Rotomotion - UAV Helicopter Controller. [Online]. Available: http://www.rotomotion.com/prd_UAV_CTLR.html

[15] Autopilot: Do it yourself UAV. [Online]. Available: http://autopilot.sourceforge.net/

[16] weControl - wePilot1000. [Online]. Available: http://www.wecontrol.ch/Flight-Control-Systems/wepilot1000-a-flight-control-system-for-unmanned-helicopters.html

[17] S. Yokum and S. Rogers, "Reconfigurable Adaptive Autopilot System for Man Portable Fixed Wing Uninhabited Aerial Vehicles," in *Proceedings of AIAA Infotech@Aerospace*, September 2005.

[18] W. Alvis, "Development of an FPGA Based Autopilot Hardware Platform for Research and Development of Autonomous Systems," Ph.D. dissertation, University of South Florida, March 2008.

[19] W. Alvis, S. Murthy, K. Valavanis, W. Moreno, M. Fields, and S. Katkoori, "FPGA Based Flexible Autopilot Platform for Unmanned Systems," in *Mediterranean Conference on Control Automation*, 27-29 2007, pp. 1 –9, mED'07.

[20] G. Cai, K. Peng, B. M. Chen, and T. H. Lee, "Design and Assembling of a UAV Helicopter System," in *Proceedings of the 5th International Conference on Control & Automation, Budapest, Hungary*, 2005, pp. 697–702.

[21] M. Dong, B. M. Chen, G. Cai, and K. Peng, "Development of a Real-time Onboard and Ground Station Software System for a UAV Helicopter," *Journal of Aerospace Computing, Information, and Communication*, vol. 4, pp. 933–955, August 2007.

[22] J. Elston, B. Argrow, and E. Frew, "A Distributed Avionics Package for Small UAVs," in *Proceedings of AIAA Infotech@Aerospace*, 2005.

[23] C. Coopmans and Y. Han, "AggieAir: An Integrated and Effective Small Multi-UAV Command, Control and Data Collection Architecture," in *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, August 30 - September 2 2009.

[24] Paparazzi Autpilot. [Online]. Available: http://paparazzi.enac.fr/wiki/Main_Page

[25] C. Coopmans, “AggieNAV: A Small, Well Integrated Navigation Sensor System for Small Unmanned Aerial Vehicles,” in *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, San Diego, California, USA, August 30 - September 2 2009.

[26] S. Ates, I. Bayezit, and G. Inalhan, “Design and Hardware-in-the-Loop Integration of a UAV Microavionics System in a Manned-Unmanned Joint Airspace Flight Network Simulator,” *Journal of Intelligent and Robotic Systems*, vol. 54(1-3), pp. 359–386, March 2009.

[27] Memec, “Virtex-4 FX12 Mini Module User Guide,” October 2005, Version 1.1.

[28] ——, “Spartan-3 Mini Module User Guide,” July 2005, Version 1.0.

[29] Xilinx, “Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors,” February 2005, XAPP623. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp623.pdf

[30] Littelfuse, “TVS Diode Applications Training,” 2007. [Online]. Available: http://www.littelfuse.com/data/en/Product_Training/TVS_Diode_Applications_Training.ppt

[31] Linear Technology, “Ceramic Input Capacitors Can Cause Overvoltage Transients,” March 2001, Application Note 88. [Online]. Available: http://cds.linear.com/docs/Application%20Note/an88f.pdf

[32] “Fundamentals of PolySwitch Overcurrent and Overtemperature Devices,” 2008. [Online]. Available: http://www.circuitprotection.com/catalog/fundamentals/PSWFundamentals.pdf

[33] Maxim Integrated Products, “Reverse-Current Circuitry Protection,” January 2001, aPPLICATION NOTE 636. [Online]. Available: http://www.maxim-ic.com/app-notes/index.mvp/id/636

[34] Analogic Tech, “Step-Down DC/DC Converter Input Ripple and Noise,” 2006, AN-112. [Online]. Available: http://www.analogictech.com/uploads/download.php?type=applicationnotes&ANID=9

[35] Texas Instruments, “ISR Input/Output Filters,” June 2000, SLTA013A. [Online]. Available: http://www.ti.com/litv/pdf/slta013a

[36] Linear Technology, “Performance Verification of Low Noise, Low Dropout Regulators,” March 2000, Application Note 83. [Online]. Available: http://cds.linear.com/docs/Application%20Note/an83f.pdf

[37] Kemet, “Piezoelectric Effect in Ceramic Chip Capacitors - What is It?” March 2000. [Online]. Available: http://www.kemet.com/kemet/web/homepage/kfbk3.nsf/vaFeedbackFAQ/ECBE48478F8CAD6285256A8700515CEC?OpenDocument

[38] H. J. Zhang, “Linear Regulator and Switching Mode Power Supply Basics,” in *Linear Technology, Embedded Systems Conference (ESC)*, April 2006. [Online]. Available: http://www.eetimes.com/electrical-engineers/education-training/tech-papers/4125762/Linear-Regulator-and-Switching-Mode-Power-Supply-Basics

[39] Linear Technology, “Minimizing Switching Regulator Residue in Linear Regulator Outputs,” July 2005, Application Note 101. [Online]. Available: http://cds.linear.com/docs/Application%20Note/an101f.pdf

[40] Maxim Integrated Products, “High-Speed, Low-Voltage, CMOS Analog Multiplexers/Switches,” March 2002. [Online]. Available: http://datasheets.maxim-ic.com/en/ds/MAX4617-MAX4619.pdf

[41] Analog Devices, “AD7980: 16-Bit, 1 MSPS PulSAR ADC in MSOP/QFN,” 2009, rev. B. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD7980.pdf

[42] Silicon Laboratories, “Improving ADC Resolution by Oversampling and Averaging,” December 2003, AN118. [Online]. Available: http://www.silabs.com/Support Documents/TechnicalDocs/an118.pdf

[43] S. W. Smith, *The Scientist and Engineer’s Guide to Digital Signal Processing*, 1st ed. California Technical Pub., 1997, ch. 3, pp. 48–60. [Online]. Available: http://www.dspguide.com/ch3.htm

[44] Texas Instruments, “Analysis of the Sallen-Key Architecture,” September 2002, SLOA024B. [Online]. Available: http://focus.ti.com/lit/an/sloa024b/sloa024b.pdf

[45] B. C. Baker, “Stop-band limitations of the Sallen-Key low-pass filter,” Analog Applications Journal, Q4 2008. [Online]. Available: http://www.ti.com/litv/pdf/slyt306

[46] Freescale Semiconductor, “Noise Considerations for Integrated Pressure Sensors,” May 2005, AN1646. [Online]. Available: http://cache.freescale.com/files/sensors/doc/app_note/AN1646.pdf?fsrch=1&sr=1

[47] Xilinx, “Spartan-3 FPGA Family Data Sheet,” June 2008, DS099. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[48] ——, “EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design,” 2009, XTP013 EDK 10.1. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/edk_ctt.pdf

[49] ——, “Embedded System Tools Reference Manual - EDK 10.1, SP3,” 2008. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf

[50] ——, “XST User Guide 10.1,” 2008. [Online]. Available: http://www.xilinx.com/itp/xilinx10/books/docs/xst/xst.pdf

[51] ——, “Synthesis and Simulation Design Guide 10.1,” 2008. [Online]. Available: http://www.xilinx.com/itp/xilinx10/books/docs/sim/sim.pdf

[52] ——, “Constraints Guide 10.1,” 2008. [Online]. Available: http://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf

[53] Mentor Graphics, “ModelSim Â® SE Userâ€™s Manual Software Version 6.5a,” 2009.

[54] Xilinx, “MicroBlaze Processor Reference Guide EDK 10.1i,” July 2008, UG081 (v9.3). [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[55] ——, “On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.10c),” August 2006, DS401.

[56] ——, “Processor Local Bus (PLB) v4.6 (v1.03a),” July 2008, DS531.

[57] ——, “Fast Simplex Link (FSL) Bus (v2.11a),” July 2008, DS449.

[58] ——, “Multi-CHannel OPB External Memory Controller (MCH OPB EMC) (v1.01a),” December 2006, DS500.

[59] ——, “OPB Interrupt Controller (v1.00c),” May 2006, DS473.

[60] ——, “OPB UART Lite (v1.00b),” October 2007, DS422.

[61] E. Hogenauer, “An economical class of digital filters for decimation and interpolation,” *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 29, no. 2, pp. 155 – 162, apr 1981.

[62] M. P. Donadio, “CIC Filter Introduction,” July 2000. [Online]. Available: http://www.mikrocontroller.net/attachment/51932/cic2.pdf

[63] Xilinx, “Using Block RAM in Spartan-3 Generation FPGAs,” March 2005, XAPP463 (v2.0). [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp463.pdf

[64] ——, “Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs,” March 2005, XAPP464 (v2.0). [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp464.pdf

[65] Maximum safe servo PWM pulse rate - DIY Drones. [Online]. Available: http://diydrones.com/forum/topics/maximum-safe-servo-pwm-pulse

[66] u-blox, “ANTARIS Positioning Engine Protocol Specification,” 2009, GPS.G3-X-03002-B.

[67] Microbotics Inc., “MIDG II Message Specification for Firmware V2.2.XXXX,” July 2008. [Online]. Available: http://www.microboticsinc.com/midg_files/MIDG%20II%20Message%20Specification%20FW2_2.pdf

[68] Wikipedia. (2010, October 6th) Calibrated airspeed. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Calibrated_airspeed&oldid=389110490

[69] *U.S. Standard Atmosphere, 1976*. U.S. Government Printing Office, Washington, D.C., October 1976. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770009539_1977009539.pdf

[70] Y. Kosuge and M. Ito, “Evaluating an a-b Filter in Terms of Increasing a Track Update-Sampling Rate and Improving Measurement Accuracy,” *Electronics and Communications in Japan (Part I: Communications)*, vol. 86, no. 10, pp. 10–20, october 2003.

[71] D. Tenne and T. Singh, “Optimal Design of a-b-(g) Filters,” in *Proceedings of the American Control Conference*, vol. 6, June 2000, pp. 4348–4352.

[72] Xilinx, “10.1 EDK - Why are floating point and/or hardware multiplier compiler flags/instructions NOT being used in my EDK application even though the features are enabled in MicroBlaze?” October 2008, AR31770. [Online]. Available: http://www.xilinx.com/support/answers/31770.htm

[73] comp.arch.fpga, “Xilinx’s microblaze hangs when a timer interrupt occurs after a "rand()" instruction,” February 2008. [Online]. Available: http://groups.google.com/group/comp.arch.fpga/browse_thread/thread/ebb20fbf8254fea3/516dd6caa4e190be